

NBSIR 75-735

An Expandable Total Energy Data Editor

Richard H. F. Jackson

Applied Mathematics Division
Institute for Basic Standards
Washington, D. C. 20234

Final

June 1975

Issued August 1976

Technical Report to
Center for Building Technology
Institute for Applied Technology
Washington, D. C. 20234

Prepared for
Department of Housing and Urban Development
Office of Policy Development and Research
Washington, D. C. 20410

NBSIR 75-735

AN EXPANDABLE TOTAL ENERGY DATA EDITOR

Richard H. F. Jackson

Applied Mathematics Division
Institute for Basic Standards
Washington, D. C. 20234

Final

June 1975

Issued August 1976

This report covers work done by the Applied Mathematics Division in collaboration with the Center for Building Technology under an inter-division agreement between the Building Environment Division and AMD during FY 74 and FY 75. The work was in support of the HUD-MIUS Total Energy project in Jersey City, sponsored by the Department of Housing and Urban Development.

Technical Report to
Center for Building Technology
Institute for Applied Technology
Washington, D. C. 20234

Prepared for
Department of Housing and urban Development
Office of Policy Development and Research
Washington, D. C. 20410



U.S. DEPARTMENT OF COMMERCE, Elliot L. Richardson, *Secretary*

Edward O. Vetter, *Under Secretary*

Dr. Betsy Ancker-Johnson, *Assistant Secretary for Science and Technology*

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Acting Director*

FOREWORD

The Department of Housing and Urban Development (HUD) is conducting the Modular Integrated Utility System (MIUS) Program devoted to development and demonstration of the technical, economic, and institutional advantages of integrating the systems for providing all or several of the utility services for a community. The utility services include electric power, heating and cooling, potable water, liquid waste treatment, and solid waste management. The objective of the MIUS concept is to provide the desired utility services consistent with reduced use of critical natural resources, protection of the environment, and minimized cost. The program goal is to foster, by effective development and demonstration, early implementation of the integrated utility system concept by the organization, private or public, selected by a given community to provide its utilities.

A major building-block of MIUS is the on-site generation of electricity with waste heat recovery, better known as "Total Energy." Although there are good environmental and energy conservation reasons for adopting Total Energy systems, specific detailed information on the costs of operating a total energy system for residential use is generally lacking. The Department of Housing and Urban Development has supported the construction of a test facility in Jersey City, N. J. to provide the information needed for evaluating residential Total Energy.

The National Bureau of Standards, under contract to HUD, is obtaining the needed data from the operating plant. Temperature sensors and fluid flowmeters measure the hot and chilled water systems and the fuel flow. System loads, climatic factors, building space heating and cooling loads, electricity and domestic hot water use, and environmental and economic data are being monitored. On completion of the first year of operation, a data base will be available for the first time on the operational capabilities, limitations and efficiency of a Total Energy system.

Under HUD direction several agencies are participating in the HUD-MIUS Program, including the Energy Research and Development Administration, the Department of Defense, the Department of Health, Education and Welfare, the Environmental Protection Agency, the National Aeronautics and Space Administration, and the National Bureau of Standards. The National Academy of Engineering has provided an independent assessment of the Program.

This publication is one of a series developed under the HUD-MIUS Program and is intended to further a particular aspect of the program goals.

Drafts of technical documents are reviewed by the agencies participating in the HUD-MIUS Program. Comments are assembled by one of the agencies into a Coordinated Technical Review. The draft of this publication received such a review and all comments were resolved with HUD.

ABSTRACT

This report documents the Total Energy Data Editor as of January 31, 1975. It is a computer program developed to process the data to be collected by the ongoing Total Energy Project at the National Bureau of Standards. Consisting of a mix of FORTRAN and RAYTHEON machine language subroutines, the Editor is a powerful, interactive program written to be run on a Raytheon 704 minicomputer with two tape drives and a disk pack. Since this document is also meant as a user's manual, it includes a dictionary of commands, complete discussions and listings of individual subroutines, as well as an explanation of the workings of the program.

Contents	Page
0. INTRODUCTION: THE TOTAL ENERGY DATA EDITOR - WHAT IT IS	1
1. THE DATA EDITOR - WHY IT IS WHAT IT IS	3
2. THE DATA EDITOR - WHAT IT DOES	4
3. THE DATA EDITOR - HOW IT WORKS	6
4. THE DATA EDITOR - HOW TO TALK TO IT	9
5. THE DATA EDITOR - HOW TO USE IT	14
6. THE DATA EDITOR - ITS PARTS (THE SUBROUTINES)	16
7. THE DATA EDITOR - HOW IT TALKS TO YOU	47
8. BIBLIOGRAPHY	50
APPENDIX A: SUBROUTINE LISTINGS	A-0
INDEX TO APPENDIX A	A-i
APPENDIX B: FORMAT OF RAW DATA INPUT TAPE	B-0
APPENDIX C: FORMAT OF 1108 COMPATIBLE OUTPUT TAPE	C-0

O. INTRODUCTION: THE TOTAL ENERGY DATA EDITOR - WHAT IT IS

This report documents the Total Energy Data Editor, as of the "freeze date" of January 31, 1975. It is a computer program developed to process the data from a very specific source, namely the data collection effort of the Total Energy Project at the National Bureau of Standards (NBS). That project is concerned with investigating the feasibility of the "total energy" concept (recycling waste heat produced by on-site generation of electricity), by studying the performance at a prototype site consisting of high and low rise apartment buildings with its own power plant in the central equipment building (CEB). Approximately 300 electronic sensors were implanted throughout the site, measuring flows, temperatures, and pressures at key points.

These measuring devices are wired into an automatic data acquisition system (DAS) which automatically scans the measuring devices and takes a reading from each one every 5 minutes. Hereafter, one of these sets of readings will be referred to as a "data scan." It is planned for the data collection effort to operate for a full year. The data collected in this manner are stored on magnetic tape and, every few days, a full tape of data is shipped to NBS, edited on the Center for Building Technology's RAYTHEON 704 minicomputer and then analyzed on the NBS UNIVAC 1108.* See appendices B and C respectively for the format of a data scan and of a record on the edited tape.

The data editing requirements of the Project reflect the fact that large quantities of data (3.2×10^7 data values in the course of the year) are to be collected, in operational rather than laboratory-like surroundings (e.g., a power plant), by very sensitive electronic equipment, which under the circumstances, must be expected to malfunction at times. In view of these considerations, it was decided to use a hands-on approach to data editing. Such an approach is clearly too expensive to execute on a large batch process machine like the 1108, but is economically feasible on the CBT-owned minicomputer. This is the background of the decision to develop the package documented below.

The Data Editor program consists of a main program and 49 subroutines, which are written either in FORTRAN or in RAYTHEON machine language. Its ability to operate in an interactive mode is provided by certain subroutines which perform instruction interpretation. An extensive set of instructions is available and provision was made for easy addition of more at a later date.** It is a very core conscious program, because throughout most of the Editor's development, the RAYTHEON had only 16,000 words of storage available. It was much later in the project that the available core size doubled to 32,000. Consequently, some of the

*A more complete description of the Total Energy Project and its analysis phase can be found in [3].

**Since the "freeze" date for this documentation, that capability has been successfully utilized by D.E. Rorrer of the Center for Building Technology.

early efforts to save core storage which remain, may seem ridiculous now. The reader is forewarned of this.

Appropriate points of discussion with regard to any computer programming system are portability and versatility. Although the Data Editor was designed for the CBT-owned RAYTHEON 704, and although much of it had to be written in machine language for reasons of efficiency, every effort was made to keep the program as general and portable as possible. The modular construction discussed in the next section was useful in this respect in that machine language subroutines were written to accomplish very specific, restricted functions. In most cases, each routine could be rewritten in another machine language or even in FORTRAN with little effort, using the subroutine descriptions as a guide. On the other hand, the Editor is quite versatile with respect to applications to different data tapes. Some changes will be required in the Phase I programs (see Section 2), but all are straightforward.

As a final note, it needs to be explained that the monitor referred to in the next section (the user of the system, if you will) necessarily will be someone familiar with the Total Energy project and with the kinds of data problems that will occur. This is necessary in the early stage of the year-long data processing effort because, as is discussed in Section 1, no one knows exactly what steps will be required in the performance of a given edit, or in what sequence the steps will be performed. As a result, the Editor was designed to be flexible and to grow through addition of new functions. As the editing process becomes more routine in time, additional functions can be added to effect repetitious behavior, and the familiarity restraint on the user could be removed.

The balance of this report consists of eight sections and three appendices. Section 1 discusses the design philosophy of the Editor. Section 2 explains the editing process by elucidating the three phases of an edit. Section 3 presents an "inside" view of the Editor, by explaining the structure of an Editor instruction and discussing how such an instruction gets translated into an action. Section 4 offers a dictionary of the available instructions. Section 5 describes what a typical edit might look like. Section 6 presents each of the subroutines in the Editor with comments on its function. Section 7 explains the error messages that can appear. Section 8 is the bibliography. Appendix A contains listings of the subroutines. Appendix B presents the format of a data scan. Finally, Appendix C contains the format of the 1108 compatible output tape.

1. THE DATA EDITOR - WHY IT IS WHAT IT IS

The design concept that was paramount in developing the Editor was flexibility through simplicity. This was required because, by the nature of the project for which the Editor was developed, many of the data problems the Editor would need to handle could not be identified in advance. Those which were classical problems in data collection could of course be anticipated and allowed for in the program design. It was recognized, however, that many others would only be discovered once the data collection, editing, and analysis processes had begun and would then require resolution in a "real-time" mode.

The Data Editor, therefore, was designed from the top down, and created from the bottom up. It was designed top-down, in the sense that the whole data editing operation was looked at from the viewpoint of the person (hereafter referred to as the monitor) who will be seated at the RAYTHEON 704 performing the editing, and each of his major processes was then broken down into its basic elements. (Those processes and their corresponding component elements will be discussed in more detail later.) Those basic elements were then programmed as individual subroutines. More subroutines which combine the basic elements were then written to perform certain elementary functions. And then even more subroutines were written, to combine these functions so as to satisfy the requirements of the edit processes. Thus a hierarchy of "levels of subroutining" was developed, starting from the most elementary. These levels of subroutining are explained in greater detail in Section 6.

It is in this structure, and most especially in the breadth of the bottom level of subroutines as well as in the simplicity of each of the members of that bottom level, that the requisite flexibility of the Data Editor resides. As the need for a new function is discovered, in most cases its accomplishment should only require combining an already existing group of subroutines. Even if additional bottom level routines are required, these are by their very nature simple and easy to develop. Also, since everything is written as a subroutine, the inclusion of each new routine (and, therefore, of the new function being provided) requires almost no re-programming.

This structure should provide effectively for real-time resolution of new, unanticipated data problems: every effort was made to provide all the basic tools (in the bottom level of routines) likely to be required, and the programming/reprogramming required to resolve such new problems should be minimal.

2. THE DATA EDITOR - WHAT IT DOES

As indicated earlier, considerable time was spent in thinking through what processes would be involved in what came to be called a "data edit." The overall view of an edit of one raw data tape of information reveals three phases.

The first phase is one in which each data scan is read; checked for physical errors (bad format, parity errors, preemptive scans in the middle of other scans, etc.); converted to binary from EBCDIC; reformatted so that all scans are of fixed length with the extraneous characters removed from each DAS channel measurement, leaving only the actual measurement; recorded in the table of contiguous times which is later used as an index into the file; and finally written onto the disk in the next sequential record location. This continues until all records on the raw data tape have been processed, or until the disk file has been filled.

If any errors appear or occur during the execution of this first phase, the automatic execution of Phase I stops and control is passed to the monitor (the person doing the editing), who then corrects the error using any of the commands designed for that purpose, and restarts the automatic execution of Phase I.

The second phase of the data edit is the least automatic of the three. It is also the most amorphous in that the monitor decides what must be done to the data that is now disk resident. For example, a pass through the complete data file (or a specific section) can be made to produce some summary statistics* on the data, or to set certain status flags. (Every data value has associated with it a status flag which provides information on the quality of the datum.) On the other hand, if the daily log from the site indicates that a certain transducer appeared to be recording erratically for a specified time, then the measurements from that transducer for the time period in question could be printed out, summarized or have its status flags set, and so on. When Phase I indicates no problems with the data, this phase can be bypassed and the monitor can proceed with Phase III.

The third phase of the edit process is the one which produces the UNIVAC 1108 compatible output tape as well as an edit summary report. There is some automatic reformatting here, as well as some checking of the data. Future plans include the installation at this point of much more error checking (e.g. a check to ensure that the times on the output tape are non-decreasing.)

The edit summary produced during Phase III includes a report on changes in status flag settings across the data records, and will eventually include such items as summaries of any interpretive comments inserted into the data stream for clarification.

* STATS, the routine that produces the statistics had not been completed by the freeze date for publication of this report. See section 6 for its design specifications.

This phase is also automatic, but again, if an error occurs, control is passed to the monitor who corrects the error, and steps the Editor through each part of the third phase process until it is "in sync," in the sense of being positioned to begin the process of transferring one data record to the 1108 tape, rather than in the middle of that process. When the Editor is in sync, the monitor can restart the automatic processing.

3. THE DATA EDITOR - HOW IT WORKS

The goal of this section is to explain in some detail the inner workings of the Total Energy Data Editor. It is intended to provide the reader with a "feel" for the subroutines in the Editor, and the manner in which those subroutines interrelate. It does this by discussing the relationship between Editor command (listed alphabetically in the next section) and internal subroutine, and also the relationships between command/subroutine and the phases outlined in the previous section.

The first item of concern to a user of the Data Editor is the structure of an Editor instruction. The discussion of how the Editor works will therefore be launched with an explanation of the command structure. The command structure can also serve as a convenient vehicle for more detailed explanations of the inner workings of the Editor.

An instruction to the Editor is composed of two basic parts: the command word, and its parameters. The parameters of an instruction can be of three types. The first kind of parameter is a "local" parameter, one that is intrinsic to the particular command word. An example of this is the "disk buffer print out" command*:

PO, 14, 29

where PO is the command word and 14 and 29 are the parameters which are transmitted to the subroutine POUTBF, and are used by that subroutine to determine which words of the buffer to print out.

The second and third kinds of parameters in an instruction are, respectively, class variables and day/time references. These appear only in commands which reference the disk-resident data file and define the "two-dimensional area of interest" (AOI) block in that file. An example of a command that requires these parameters is the "set data value" command:

SD, 19, X9, 281, 1905, 282, 0145

where 19 is a local parameter, X9 is a class variable (explained below), and the rest are the day and time references (also explained below.)

Before proceeding further, the concept of a two-dimensional area of interest should be explained. It is convenient here to view the disk resident data file as a mass of data records, each 640 words long (consisting of 6 words of label information followed by 317 measurement values followed by 317 status flag words), all stacked one beneath the other in space. The up and down dimension is provided by the time

*See Section 4 for a detailed explanation of the commands referenced in this section.

associated with each scan. For example, in the above instruction, the AOI block is defined to begin with the scan made on day 281 at 1905 hours, and to end with the scan made on day 282 at 0145 hours. But this only covers one dimension of the AOI block. Since it is not expected that one would want to change every data value in a particular scan, the concept of a class variable defining the other ("across the top") dimension was introduced. A class variable is simply a variable that has been defined (using the CV command) to represent a set of measurement variables (or rather one or more of the 634 positions in the disk-resident scan). More on class variables can be found in the explanation of the CV command in Section 4.

Therefore, in the SD instruction given as an example above, the AOI block is defined in the day-time dimension by 281,1405 and 282,0145, and in the measurement variable dimension by the set of positions represented by the class variable X9.

With the format of an Editor instruction presented, the next aim is to describe what happens inside the editor to that instruction, from the time it is typed in until the time the operation is complete.

The characters representing an instruction are read in by the MAIN program in Hollerith format, converted to a special internal format by a call to subroutine CODIT and the buffer containing those characters is passed to subroutine GETCOM, which is concerned solely with instruction interpretation. The characters in the instruction are processed one-by-one and each element of the instruction is built up. The characters in the command are grouped together, and a function* is applied to the bits which make up this grouping to produce an index into the symbol table controlled by subroutine HASH. The parameters in the instruction are then processed one-by-one, independent of their meaning in the sense that if the parameter is a number, the Hollerith characters representing it are converted to binary digits and the number is built up digit-by-digit. If the parameter is a variable (see the SV and DV commands in Section 4), its characters are also combined in the same sense that the command characters were, and a hashing function is also applied to yield an index into the symbol table, where the value of that variable is retrieved. This process of parameter processing continues until all are processed.

At this point, control, as well as the buffer containing the interpreted parameters, is passed back to MAIN which obtains the "command number" from the symbol table, using the hashed command characters as an index. This command number is then used in a computed GO TO statement to jump to a subroutine CALL statement, which begins the next part of the process.

If the command was one of those that do not deal directly with the disk data file, and consequently do not have the AOI block parameters as part of the command, then this part of the operation consists simply of a

*Functions such as these are commonly referred to as hash functions.

CALL statement directly into the subroutine which performs the function requested; e.g., an RT command causes a direct call to the RDTAPE subroutine.

If on the other hand, the command does have the AOI block parameters in it and, therefore, requires access to a number of data scans on the disk, control passes first to subroutine DRIVER. This subroutine's main role is to interpret the AOI block and retrieve, in succession, the data scans requested. It does this by computing the relative position of each scan in the data file, using information stored in TABLE, the table of contiguous time blocks which is maintained by subroutine UTABLE. Having determined the position in the file, it causes the record to be read in by a CALL to DSKRTS. Once the record is read in, DRIVER passes control to the subroutine which is to perform the desired action; e.g., STDATA to set data or status flag values, or DATTAP to reformat and output the record to the 1108 compatible tape. When the subroutine finishes operating on the one record, it sends control back to DRIVER which writes the edited scan back onto the disk, if necessary, gets the next record needed, and continues on.

That, in brief, is the flow through the Editor as experienced by an instruction statement as it gets passed around. A more detailed understanding of the individual subroutines can be obtained by reading the specification sheet for an individual subroutine given in Section 6 or by reading the listings which appear in Appendix A.

4. THE DATA EDITOR - HOW TO TALK TO IT

This section contains a dictionary of the Editor commands that had been built in by the cut-off date for inclusion in this report. As mentioned earlier, the Editor was designed to be dynamic and constantly growing, in order to be able to resolve the new data problems as they arise. This of course means that the list is not all inclusive, even as of the date of this report; it is expected that updates will be issued periodically.

Section 3 of this report gave some information regarding the general format of an instruction to the Data Editor. In that section, the concept of a command word and its associated local and AOI block parameters was introduced. It should be noted that the exact format of an instruction is the command word followed by a comma, followed by the local parameters and then the AOI block parameters, all separated by commas. Within this overall structure, an instruction is free form in that imbedded blanks are ignored and a comma followed by a comma or by a series of blanks and then a comma indicates a parameter with a value of zero. There is no end of line character. Nor is there a line continuation character. The import of the last comment is that all instructions must fit on one 72 character line of the CRT. This affects only the CV instruction, and special provision was made to handle more input for that instruction.

The day-time references mentioned in Section 3 and referred to below as D1,T1 and D2,T2 have a special format: day (D1 and D2) is given as day of the year, and time (T1 and T2) is given in the form of the 24 hour clock. Thus, 9:15 PM November 18 would be input as 322, 2115.

Another point to mention is that by setting system flag 10 on the RAYTHEON 704 before loading the Data Editor, all output written to the CRT will be copied automatically when the CRT face becomes filled, provided that the CRT function switch is turned on. When the automatic copying is complete, the face is erased and outputting continues.

A last point before providing the dictionary is that all instructions in the dictionary are mnemonic. In what follows, the phrase from which the mnemonic is formed is given, in most cases, in the first sentence of its definition with the appropriate letters underlined.

* * *

BK,N,L

This command will cause logical unit L to be backspaced over N records. If N is negative, the unit will be skipped forward.

C8,D1,T1,D2,T2

Create the 1108 tape. This causes all scans from day D1, time T1 through day D2, time T2 to be reformatted and written, along with their status flags, onto the output tape. The data scans copied are currently residing on the disk; and the output tape is mounted on the tape drive assigned to logical unit 9. This command also checks the status flag settings across the scans. If any settings change from one scan to the next, that information is printed out onto the listing device (logical unit 11), which is normally the CRT.

CN

Converts, checks, and reformats the data scan currently stored in the input buffer, and then rewrites it into the output buffer. The data scan is converted from individual EBCDIC characters to individual binary representations of single digits, then grouped into binary values which represent the measured value, (checked during this process for character legality), and finally stored in the proper place in the output buffer. If any illegal characters are encountered during the conversion, a message to that effect is printed on the CRT.

CV,A,S,N1,N2,...,N25

This is the class variable processing command. A class variable is one that is used to represent a number or set of measurement variables. The set could be defined to be all temperature measurements or all pressures, for example. The representation is by position in the disk data record and is defined, redefined and "undefined" with the CV command.

Here, A is the name of the variable being referenced or defined. It can be any two characters from the set of letters and numbers, but the first must be a letter and the resulting name must be different from the command names as well as all previously defined variable names. S in the command is a switch. If S=0, this class variable is being defined to represent the measurement variables occupying the (not necessarily contiguous) positions N1,N2,...,N25. If S<0, the class variable A will be deleted from the class variable table. And if S>0, class variable A will be expanded to include the additional positions N1,N2,...,N25 in its definition along with its previous values. It should be noted that 25 is the upper limit on the number of positions in one CV command only because one line of input from the CRT can contain no more than 72 characters and there is no provision for continuation lines.

DB,N

This command causes the value of the internal variable DEBUG to be set to the value N. If N>0, the debugging prints associated with subroutine N will be printed each time that subroutine is entered. The subroutine numbers are:

1. MAIN	2. GETCOM
3. HASH	4. INHASH
5. DEHASH	6. INIT
7. CLASS	8. COMINT
9. ERRPRT	10. RDTAPE
11. CONALL	12. POUTBF
13. HEXDMP	14. UTABLE
15. DRIVER	16. DSKRTS
17. DATTAP	18. EX
19. STDATA	20. CVPROC

A value for DEBUG of zero, of course, turns off all debugging prints.

DV,A

Deletes a variable A and its associated value from the symbol table. See the SV command for more.

EX

This command terminates an editing run and exits to the RAYTHEON system. Before terminating the run, it performs a clean-up function by tying off all loose ends in the editing run (e.g. the closing out of the disk files). Current plans include extending this routine to allow for storing enough information to provide a restart capability if it becomes necessary to abort an edit run before completion.

GS,N

Gets the data scan that occupies the N'th position (starting from 1) on the disk file, and copies it into the disk buffer. Note that this works strictly by position on the disk, not by day and time.

HD,N1,N2

Provide a hexadecimal dump of the characters numbered N1 through N2 in the raw data buffer.

LS,N1,N2,N3

Left shifts by N1 positions, the characters from N2 through N3 of the raw data buffer. This command also zero fills the positions shifted out of. CAUTION: It is extremely important not to shift beyond the limits of the buffer, since this will destroy other areas of core with untoward consequences.

NV,N

Sets the value of the internal variable NVAR to N. NVAR represents the number of DAS channels (measurement variables) that were scanned and subsequently recorded on the raw data tape that is about to be processed. If set too large, error messages will appear the first time a record is read. If set too small, data will not be transferred and will thus be lost during the transferring in a RT instruction.

PO,N1,N2

This command is used to print out the contents of the disk buffer. The label characters are printed first, followed by the values in positions N1 through N2. If N2 is 0 or does not appear, just position N1 is printed. If neither N1 nor N2 appear, or are both 0, only the label is printed on the CRT.

RS,N1,N2,N3

Right shifts by N1 positions, the characters from N2 through N3 of the raw data buffer. See the discussion of the LS command for more.

RT,N

Reads one record from the raw data tape and copies into the raw data buffer the data in its original unconverted form (EBCDIC characters - 13 per DAS channel). N is the number of characters in the record to skip before beginning the transfer.*

RW,L

Rewinds logical unit L.

SD,V,A,D1,T1,D2,T2

This command provides the capability for setting the value of data words or their associated status code words. In the command string, V is the value the words will be set to; A is either a number representing the position in the data scan to be reset, or a class variable (previously defined) representing a set of positions to be all set to V; D1 and T1 are the day and time of the first data scan on the disk to be thus set; and D2,T2 are the day and time of the last disk-resident data scan to be thus set.

SE,N,L

Positions logical unit L past the N'th end-of-file. A message is printed indicating the number of records skipped (including the ends-of-file). The mnemonic is from "seek end-of-file".

SK,N,L

Positions logical unit L beyond the N'th record. The mnemonic is from "skip a record".

SP,N

Sets to N the update pointer word in the I/O control block of the File Control System (FCS). When used in conjunction with the US command, it allows the writing of the contents of the disk buffer to a specified record position on the disk.

*Since the freeze date for this report, the option to skip characters before reading has been installed in a separate command.

SV,A,N

Creates a variable, A, and sets its value at N. Any subsequent commands in which A appears as one of the arguments will have the value of A (N in this case) substituted for it. Restrictions on the variable name A, are the same as those described under the CV command.

TR,N,K

Transfers raw data to the disk by traversing, N times, the loop of reading a raw data record from the data tape, converting, checking, regrouping, reformatting, indexing, and finally writing the information on the disk. After every k records are transferred in this manner, the command produces a message on the CRT indicating this fact.

US

Updates a scan by writing the data scan that currently resides in the disk buffer on the disk as the N'th record on the disk, where N is the current value of the disk update pointer in the FCS I/O control block. The update pointer is set explicitly by using the SP command. It can also be set implicitly by the GS command, since every time the GS command is executed, the update pointer is automatically set to the position of the record just transferred.

WE,L

Writes three ends-of-file on logical unit L.

WS

Write the data scan currently disk buffer resident onto the disk in the next available record position (usually at the end of the file).

5. THE DATA EDITOR - HOW TO USE IT

In this section, the reader is presented with a typical edit process. The intent is to provide an understanding of how everything fits together into a cohesive system which performs the desired data editing function. No attempt will be made here to explain how to operate the RAYTHEON 704. The Data Editor is a system resident routine on that machine, and the monitor will load the program before the editing process begins.

The editing process begins with mounting the raw data tape on the 9 track tape drive. Having done so, the monitor responds to the Editor's instruction query (a "?") with a TR command to begin the transfer of data scans from the tape to the disk. If any errors occur, the program prints error messages and waits for the monitor to decide what to do about the errors. If the errors are of the illegal-character-type, the monitor can attempt to fix the characters by dumping the contents of the EBCDIC input tape buffer to determine what the character should be, and then setting the proper value with the SD command. If it is impossible or deemed improper (in order to maintain the integrity of the data) to fix data values in this way, the monitor can simply use the SD command to turn on the status flag bits associated with the measurement containing the illegal character. Having done this, the monitor can restart the automatic transfer by stepping the editor through the completion of the transfer loop (using combinations of the RT, CN, and WS commands) until it gets back "in sync", and then by giving another TR command.

Another problem that can occur at this point is the discovery of a preemptive scan. A preemptive scan is one that results from some alarm condition at the site and might override the recording of a regular scan, with the result that the preemptive scan starts in the middle of the regular one and the tape record is larger than normal size. If this error occurs, the monitor will fix it by using the RS command to move the preemptive scan characters out of the normal scan and using the WS command to store the regular one. Then, by using the BK command to backspace over the record and the RT command to skip over the good scan characters and fill the buffer with just the preemptive scan characters, the monitor can reorganize the records so that they conform to format. Then the Editor is stepped through the transfer loop as outlined above and restarts the process with another TR command.

Once all the records have been transferred to disk, the Phase II perusal process begins. Any flags or data values that need to be set, will be set by using the SD command. Anecdotal comments are inserted, statistics gathered, and data values perused for correctness.

Once all is considered well with the data, the monitor will give a C8 command. This begins the process of transferring the scans to the 1108 compatible tape which is normally mounted on the 7 track tape drive. The only error messages that could appear will have to do either with physical tape problems (which require mounting a new tape and restarting this phase),

or with out-of-sequence scan times (which require retyping the C8 command). When the data have all been reformatted and transferred to the output tape, the WE command is used to endfile the tape. The EX command is then given to get out of the program and the day's edit is complete.

6. THE DATA EDITOR - ITS PARTS (THE SUBROUTINES)

This section describes the individual subroutines that comprise the Data Editor, each one in turn, in a "specification sheet" manner. There is an entry for each subroutine in the system, as well as for routines that have not been written as of the date of this report. These last are part of the overall design of the Editor, and as such are intrinsic to any discussion of the work done on it. These unfinished subroutines appear below with the word "FUTURE" after their names. They are left as future work on the Total Energy Project.

In an attempt to provide somewhat of an overview, we first present here a list of all subroutine names, grouped as to whether they are involved in instruction interpretation, or are level 1, 2 or 3 routines (see section 2). The names marked with an "*" are machine language routines written by D. E. Rorrer of CBT.

LIST OF SUBROUTINES

Instruction Interpretation

MAIN	GETCOM	READIN*
------	--------	---------

Level 3

CR8TAP-FUTURE	DRIVER	TRNRAW
---------------	--------	--------

Level 2

ADDTIM-FUTURE	CONALL	CVPROC
DATTAP	DSKRTS	EX
GENTXT-FUTURE	INIT	PDISK-FUTURE
RDTAPE	STATS-FUTURE	STDATA
UTABLE		

Level 1

ADRESS*	BACKSP*	CLASS*
CLOSE*	CODIT*	COMINT*
CONATE*	CONETA*	CONGRP*
CREATE*	DECDIT*	DELETF*
DEHASH	DSKINT*	DSKRD*
DSKUPD*	DSKWT*	ERRPRT
HASH	HEXDMP	INHASH
LSHIFT*	OPENIT*	OUTHEX*
OUTINT*	OUTTXT*	POUTBF
READMT*	RSHIFT*	RWNDIT*
SEEKEF*	SKIP*	SPLITA*
UNSPLT*	WRITEF*	WRITMT*

ADDTIM-FUTURE

FORTRAN REFERENCE:

CALL ADDTIM(PAR,MPA)

FUNCTION:

To place a new group of time entries on the KALNDR for a record that had been created previously by one of the non-data record generators.

ERROR CONDITIONS:

Out of space in KALNDR.

COMMENTS:

KALNDR is a linked list which stores, linked by time, a file reference number and a time associated with that file. The file referenced is a file of anecdotal or calibration information which is to be included on the output tape just before the data scan whose time is the same as that stored in KALNDR. Those "non-data" files are created by GENTXT.

ADRESS

FORTRAN REFERENCE:

CALL ADRESS(VAR,IADR)

FUNCTION:

To retrieve the absolute address of a variable or subroutine entry point.

ERROR CONDITIONS:

Variable must be defined within FORTRAN.

COMMENTS:

Either one or both of the arguments can be array elements. This is a machine language subroutine.

BACKSP

FORTRAN REFERENCE:

CALL BACKSP(N,L)

FUNCTION:

To backspace N records on unit L.

ERROR CONDITIONS:

Load point reached before completing the command.

COMMENTS:

If N<0 on output, the load point was reached. If N 0 on input, this subroutine is equivalent to the SKIP subroutine. In any case on return, |N| is the number of records skipped over. This routine was written in machine language so that it would not be necessary to include the RAYTHEON's FORTRAN tape library routines which use an amount of core storage inordinate for this application.

CLASS

FORTRAN REFERENCE:

KARCLS = CLASS(K)

FUNCTION:

This integer function subroutine is used by GETCOM to determine the character class of a given character, for use in the finite state transition table, (presented in the documentation of subroutine GETCOM), which governs the decoding of an Editor instruction.

ERROR CONDITIONS:

Only as given below.

COMMENTS:

CLASS (an subsequently KARCLS in the statement above) will be set as follows:*

CLASS = 1, if $10 \leq K \leq 35$,	(letters);
CLASS = 2, if $0 \leq K \leq 9$	(digits);
CLASS = 3, if $K = 36$,	(comma);
CLASS = 4, if $K = 41$,	(\$);
CLASS = 5, if $K=38$ or $K=39$,	(+ or - signs);
CLASS = 6, otherwise	(error).

This routine is written in machine language.

CLOSE

FORTRAN REFERENCE:

CALL CLOSE(IOT,L)

FUNCTION:

To close out the file referenced in the file control block (IOT).

ERROR CONDITIONS:

As noted below.

COMMENTS:

This is a machine language routine that sets up to call the CLOSEFL routine of the RAYTHEON disk File Control System (FCS). IOT is the Input/Output table described in the documentation of FCS, and L is a status word set by this routine. If $L > 0$ the operation is complete, no error occurred and L contains the number of words transferred. If $L < 0$, an error occurred, and the absolute value of L corresponds to the error codes given in the FCS documentation (see p. 39 of [1]).

CODIT

FORTRAN REFERENCE:

CALL CODIT(IBUF1,IBUF2,N)

*See the documentation for subroutine CODIT for more on the internal coding.

FUNCTION:

To convert the N characters, stored two to a word in IBUF1, from ASCII to the Editor's internal code and store them one to a word in IBUF2.

ERROR CONDITIONS:

None, since illegal characters have a code and are dealt with elsewhere.

COMMENTS:

The Editor's internal code is:

0 through 9	are	0 through 9,
A through Z	are	10 through 35,
,	is	36,
blank	is	37,
+	is	38,
-	is	39,
*	is	40,
\$	is	41,
all others	are	42.

Note that IBUF2 must be twice the size of IBUF1. The two buffers may be the same, in which case no more than half of the buffer may contain data. This machine language routine was written at a time when both a PDP-10 and the 704 were being used for development of the Editor and it was necessary to create a machine independent internal representation.

COMINT

FORTRAN REFERENCE:

CALL COMINT(IBUF,N,IERR,IVAR)

FUNCTION:

To convert the N digits stored in separate contiguous words of IBUF into one binary number, and store it in IVAR.

ERROR CONDITIONS:

If IVAR>32767 or IVAR<-32767, then IERR=99. If an illegal character is encountered in IBUF, IERR is set to that character. Otherwise, IERR=0.

COMMENTS:

This is a machine language subroutine.

CONALL

FORTRAN REFERENCE:

CALL CONALL(INBUF,MIN,OUTBUF,MOU,IERR)

FUNCTION:

To convert the data scan stored in INBUF from EBCDIC characters to individual binary values and store them in OUTBUF.

ERROR CONDITIONS:

None that are intrinsic to CONALL. The errors that can occur and are printed out as a result of a call to ERRPRT from CONALL, are errors that occur in the subroutines that CONALL calls. Those errors are explained in those subroutines or in Section 7.

COMMENTS:

This is the routine one calls by giving the editor a CN command. It is a level 2 routine and functions by setting up arguments required by the level 1 routine it calls which do the actual work of converting.

CONALL operates first on the 19 EBCDIC characters that form the header. These are converted into 19 binary values and stored in OUTBUF by a CALL to CONETA. The first 12 digits representing the label are then combined into 3 binary values by 3 CALL's to COMINT. The 3 digits representing the day are then combined into one value and stored in OUTBUF by another CALL to COMINT. Next, the same is done to the 3 digits representing time. Finally, CONALL causes each of the 13 character measurement groups to be converted and combined into the 5-value measurement groups that appear in the output buffer, OUTBUF. The conversion of the measurement groups is effected by CALL's to CONGRP.

CONATE

FORTRAN REFERENCE:

CALL CONATE(IBUF1,IBUF2,ICNT,IERR)

FUNCTION:

To convert ICNT ASCII characters from IBUF1 to EBCDIC and store them in IBUF2.

ERROR CONDITIONS:

IERR will be set to the number of illegal characters encountered. Each such character will be set, in IBUF2, to a zero.

COMMENTS:

IBUF1 and IBUF2 may be the same buffer, but in any case are single integer arrays with the data packed two characters per word. This is a machine language routine.

CONETA

FORTRAN REFERENCE:

CALL CONETA(IBUF1,IBUF2,ICNT,IERR)

FUNCTION:

To convert ICNT EBCDIC characters from IBUF1 to ASCII and store them in IBUF2.

ERROR CONDITIONS:

IERR will be set to the number of illegal characters encountered. Each such illegal character will be set to an ASCII "*".

COMMENTS:

This is simply the reverse of CONATE, and all comments there apply here. This subroutine is written in RAYTHEON 704 machine language.

CONGRP

FORTRAN REFERENCE:

CALL CONGRP(IBUF,IGRP,N,IFLAG)

FUNCTION:

To convert the 13 EBCDIC characters, beginning with the Nth character, of IBUF, into 5 distinct binary values and store them in IGRP.

ERROR CONDITIONS:

The 4th through the 16th bits of IFLAG will be set according as any of the Nth through the (N+ 12)th characters of IBUF are illegal. IFLAG=0 means no errors. If an illegal character appears, the corresponding word of IGRP will be set to negative zero (1 000 000 000 000 000).

COMMENTS:

The values stored in IGRP are as follows:

IGRP(1) = scanner channel number;

IGRP(2) = remote number if data was from a remote, otherwise = -1,

IGRP(3) = the DVM function (M or V);

IGRP(4) = the measurement value (including sign over-range flag and voltage digits); and

IGRP(5) = the scale digit.

for more on this, see the listing in the appendix.

This is a machine language routine.

CR8TAP-FUTURE

FORTRAN REFERENCE:

CALL CR8TAP(PAR,MPA,TABLE,MTA1,MTA2,OUTBUF,MOU,CLSTAB,MCL)

FUNCTION:

To do everything necessary to transfer from disk and core to the 1108 compatible tape all information (at this point edited and clean) associated with the time interval defined in PAR. It will also have dominion over the production of the edit summary.

ERROR CONDITIONS:

The primary CR8TAP error condition would be an overlap of the time interval requested with the already-tape-resident time interval. There are, of course, many other error conditions that could occur during the run of this routine, but they occur in lower level routines called by CR8TAP and are discussed in the documentation of those routines.

COMMENTS:

This is a level 3 subroutine (see section 2) and is consequently one of the more powerful (and important) ones. Its method of operation is first, to check KALNDR for the first non-data record to be transferred to tape. It will then have all data records with time-of-scan values less than the time of that KALNDR entry transferred to the tape. (DATAP, the routine called by CR8TAP to effect this, also produces a summary report on status flag changes over the records it transfers.) As each non-data record is written onto the tape, it is also printed out as part of the edit summary. This processing cycle continues until all information associated with the time interval stated in the instruction (stored in PAR) has been transferred.

Upon completion of the transferring function, CR8TAP will then move into its clean-up phase, wherein all core storage areas left behind must be cleaned up and "freed" for future use by the Editor. The KALNDR and possibly the non-data files must be cleaned up. It is also possible, yet unclear this point that TABLE, the time table of contiguous times (see UTABLE write-up) and the data files need to be cleaned up and their space made available.

CREATE

FORTTRAN REFERENCE:

CALL CREATE(IOT,L)

FUNCTION:

To create a disk file as described in the file control block, (IOT).

ERROR CONDITIONS:

As described in documentation of subroutine CLOSE.

COMMENTS:

This is a machine language routine that provides the linkage to get to FCS's routine CREATEFL. See subroutine CLOSE for more.

CVPROC

FORTTRAN REFERENCE:

CALL CVPROC(PAR,MPA,CLSTAB,MCL)

FUNCTION:

To define, delete, or redefine a class variable. PAR contains the parameters from the CV instruction (see section 4) and CLSTAB is the class variable table.

ERROR CONDITIONS:

There are three error conditions, each with an associated error print. They correspond to error types 17,18, and 19, and are defined in Section 7 (q.v.).

COMMENTS:

PAR(3) corresponds to S in the CV instruction (see section 4). It is -1,0, or 1 according to whether one wants to delete, define, or redefine a class variable. PAR(2) contains the encoded variable name* which is used as an index into the symbol table controlled by subroutine HASH. The value of the class variable in the symbol table is a pointer into CLSTAB to retrieve the set of measurement variable positions that the class variable represents. PAR(4) through PAR(MPA) contain the positions. Pictorially, the class table (CLSTAB) is:

symbol table index	N1	V1	V2	...	VN1	symbol table index	N2	V1	V2	...	VN2	...	-1
--------------------------	----	----	----	-----	-----	--------------------------	----	----	----	-----	-----	-----	----

where "symbol table index" is PAR(3) above, N1,N2, etc. are in each case the number of members of the class, and V1,V2, etc. are the positions. There is a -1 at the end of it all.

A class variable is defined simply by adding its information to the end of CLSTAB beginning with the position occupied by -1. A deletion is performed by removing the cells allocated to a class variable and repacking the table. A redefinition is performed by adding the additional positions to the end of a class variable definition, and for that reason can only be done on the class variable that is occupying the last physical positions in CLSTAB. If it is desired to redefine one that is not last in the list, the user must first delete the variable and then start all over.

*For more on the encoding, see the CODIT and GETCOM write-ups.

DATTAP

FORTRAN REFERENCE:

IERR = DATTAP(PAR,MPA,OUTBUF,MOU,CLSTAB,MCL,WRTSWT,LRECF)

FUNCTION:

To transfer the data record in the disk buffer (OUTBUF) to the 1108 compatible tape. Also, to produce a local edit summary consisting of changes in status flag settings from the previously transferred data record.

ERROR CONDITIONS:

Error returns from ~~WRITMT~~ routine which signal an unrecoverable write error on the output tape.

COMMENTS:

The status flag check is performed by a bit-by-bit logical OR over all the status flags in the scan. This "sum" is compared against the same value for the previous scan sent out. If different, a message is printed stating that fact. Of course, this is a valid check only if the flags are stored in independent positions in the status flag word.

DECDIT

FORTRAN REFERENCE:

CALL DECDIT(IBUF1,IBUF2,N)

FUNCTION:

To convert the N characters stored one to a word in IBUF1 from the internal code back to ASCII and pack them two to a word in IBUF2.

ERROR CONDITIONS:

None

COMMENTS:

The internal code is explained in documentation for subroutine CODIT. This is just the reverse of that routine. IBUF1 and IBUF2 may be the same storage area. This is a RAYTHEON machine language routine.

DELETE

FORTRAN REFERENCE:

CALL DELETE(IOT,L)

FUNCTION:

To delete the file described in the file control block (IOT) from the disk, and free its space for further use.

ERROR CONDITIONS:

See CLOSE documentation.

COMMENTS:

This is just an interface routine to get to FCS's routine DELETEFL. See CLOSE documentation for more.

DEHASH

FORTTRAN REFERENCE:

CALL DEHASH(KEY)

FUNCTION:

To delete a variable name and value from the symbol table controlled by the HASH subroutine.

ERROR CONDITIONS:

KEY request not found in symbol table.

COMMENTS:

This KEY is the very same one used by HASH as an index into the symbol table.

DRIVER

FORTTRAN REFERENCE:

CALL DRIVER(FUNC,PAR,MPA,TABLE,MTA1,MTA2,OUTBUF,MOU,CLSTAB,MCL)

FUNCTION:

To serve as a controlling point for all routines that require access to the disk stored data scans. It performs all operations required to translate the day and time requests (see section 4 on the structure of a command) stored in PAR, into disk accesses which bring the desired records into core one at a time. Upon this, control is transferred to the subroutine name received in argument FUNC.

ERROR CONDITIONS:

Error status is controlled by subroutines that DRIVER calls. If one of these returns in an error condition, DRIVER closes out the file and passes control to MAIN which then calls for another instruction. Disk operation errors could also occur, but these are explained elsewhere. A last error condition that DRIVER contains is one where the requested times are just unknown.

COMMENTS:

TABLE is the table of contiguous times (see subroutine UTABLE), OUTBUF is the disk buffer, and CLSTAB is the class variable table (see CVPROC documentation). Note that those arguments required by the routine that DRIVER will call, must be passed into DRIVER, so that if new subroutines are added that will deal with disk stored data scans, and these routines require parameters not already in DRIVER's calling statement, then DRIVER must have its SUBROUTINE statement expanded.

DRIVER maintains a logical variable which tells it whether to update the disk scan that it had read and passed to the subroutine requested. This switch (WRTSWT) is set in the called routine.

DRIVER also maintains a switch (LRECF) which signals that the currently stored scan is the last one requested, so that the called routine can finish out its operation if necessary (e.g. STATS).

Note that DRIVER passes off to its routines by a FUNCTION subroutine reference. Due to the vagaries of the RAYTHEON 704 FORTRAN compiler, this is the only way to pass the name of a subroutine as an argument in a subroutine CALL.

A future addition to DRIVER will be one to provide for storing the day-time and class variable requests that appear in PAR, so that once input, they can be optional in subsequent commands if the same ones are desired.

DSKINT

FORTTRAN REFERENCE:

CALL DSKINT(IOT,L)

FUNCTION:

To provide the linkage to FCS's routine INITLIZE which prepares the disk for use.

ERROR CONDITIONS:

See documentation for subroutine CLOSE.

COMMENTS:

This routine also completely cleans out the disk files there, so that all space is ready for use. Note that this is a function of the linkage routine and is not a service provided by FCS. For more, see subroutine CLOSE.

DSKRD

FORTRAN REFERENCE:

CALL DSKRD(IOT,L)

FUNCTION:

To link up with FCS's routine GETREC which brings a record, described in the file control block IOT, into a buffer (in the Data Editor's case, OUTBUF).

ERROR CONDITIONS:

See write-up of CLOSE.

COMMENTS:

The 27th word of IOT is the one that tells FCS which block on the disk to retrieve. FCS starts counting blocks from zero, so that if one wants the first record that was written to the disk then IOT(27) = 0, the second record requires IOT(27) = 1, etc. See CLOSE for more.

DSKRTS

FORTRAN REFERENCE:

CALL DSKRTS(OUTBUF,MOU,N,ICOM,IERR)

FUNCTION:

To provide a central routine through which to read, write, or update a disk record.

ERROR CONDITIONS:

Error returns from DSKRD, DSKWT, DSKUPD, which force an error print from this routine.

COMMENTS:

In a system as complicated as the Data Editor, it makes good programming sense always to open, operate on one record within, and then close out a disk file rather than to open it once in the beginning of processing and close it at the end of a day's editing. This principle and the nature of the RAYTHEON's disk File Control System, necessitated some seemingly redundant bookkeeping in the I/O control blocks (IOCB). Rather than have that bookkeeping appear everywhere a disk operation was required, this level 2 routine was written to centralize the disk accessing. See CLOSE for more.

DSKUPD

FORTRAN REFERENCE:

CALL DSKUPD(IOT,L)

FUNCTION:

To provide the linkage to FCS's routine UPDATE, which updates (rewrites, or overwrites) the record on the disk that is described in the file control block (IOT).

ERROR CONDITIONS:

See CLOSE documentation.

COMMENTS:

This is the only way to write a record to disk in other than the next available location at the end of the file. The pointer used by DSKUPD (or UPDATE in FCS) is IOT(31). It is automatically set when a DSKRD (or GETREC in FCS) operation is performed, so that one can read, revise, and immediately rewrite, without having to set anything. But one can also set the pointer independently. See CLOSE for more.

DSKWT

FORTRAN REFERENCE:

CALL DSKWT(IOT,L)

FUNCTION:

To link to FCS's routine PUTREC which writes the contents of the buffer referred to in the file control block (IOT) into the next available location in the disk file that is also described in IOT.

ERROR CONDITIONS:

See CLOSE write-up.

COMMENTS:

This routine can only write to the next available space. The PUTREC pointer described in FCS manual (see p. 26 of [1]) is in error. It is not possible as indicated there, to PUT a record wherever you want by setting the PUTREC pointer.

For more on this routine, see CLOSE.

ERRPRT

FORTRAN REFERENCE:

CALL ERRPRT(ISUB,MESS,IARG1,IARG2,IARG3,IARG4,IARG5)

FUNCTION:

To print an error message.

ERROR CONDITIONS:

None

COMMENTS:

The components of an error message are: the name of the subroutine in which the error occurred, the type number of the type of error, and the arguments that were relevant to the error. The subroutine name is stored as Hollerith characters and ISUB is a pointer to the name needed. MESS is the number of the message (see the appendix). If the error was a disk error, ERRPRT also produces a hexadecimal dump of the disk I/O control block. The arguments through IARG4 are printed in integer form and IARG5 is dumped in hexadecimal format.

EX

FORTTRAN REFERENCE:

CALL EX

FUNCTION:

To "clean up" and exit to the system.

ERROR CONDITIONS:

None as yet.

COMMENTS:

At present, almost no cleaning up is done by EX. Future plans include options such as the writing out onto disk of the I/O control blocks used to access the disk, the time table used as an index into the data scans, perhaps even the class variable table (CLSTAB) and the hash table (KEYS and VALUES). These and others would be stored so that one could reenter the Editor in a restart or continuation mode.

GENTXT-FUTURE

FORTTRAN REFERENCE:

CALL GENTXT

FUNCTION:

To add a file of anecdotal information to the non-data area of the disk.

ERROR CONDITIONS:

No more disk space left for such information.

COMMENTS:

These anecdotal comments are statements, usually copies of entries from the site log that would be valuable to subsequent researchers looking at the data. They might be comments to the effect that a particular measuring device (e.g. thermocouple no. 634) was taken out and replaced with a new one at 1345 on day 319. This information is then associated with a time on KALNDR through the ADDTIM subroutine and merged

in the output tape at its appropriate place by the CR8TAP routine.

GENTXT works by first creating a file, then switching into input mode where it requests one line of text at a time from the CRT. It continues in this fashion until the disk buffer is full, whereupon it writes that buffer-full into the file, and starts over requesting input. This cycle continues until a line with naught but END in cols. 1-3 is encountered, whence it pads out the remainder of the buffer, writes it out and sends the message containing the file reference number, to be used in the ADDIIM subroutine, back to the CRT.

GETCOM

FORTTRAN REFERENCE:

CALL GETCOM (COMMND,MCO,FCT,PAR,MPA,FLAG)

FUNCTION:

GETCOM is the heart of the interface between the user's typed-in instructions and the Data Editor's subroutine calls. It's function is to get the command instruction stored as alpha- numerics in COMMND, and convert it into a subroutine index number, stored in FCT, and a set of subroutine parameters, stored in PAR.

ERROR CONDITIONS:

Reference in the instruction string to an unknown variable, and incorrect instruction structure. This last is explained in detail below, but one example is discovering a sign (+ or -) embedded in a string of digits. In either of these cases, FLAG is turned on.

COMMENTS:

GETCOM decodes and interprets the instruction string, character by character, through the use of a finite state transition table. As each character in turn is removed from the strings, its character class is determined by a call to CLASS. Given the class of the character at hand (a number from 1 to 6), and the current state of the decoding process (a number from 1 to 9), the two dimensional finite state transition table yields the new state of the process. In other words, knowing where we came from and what type of character we have, the table tells us what to do next.

There are 10 states that GETCOM can be in:

- 1.) Idle,
- 2.) Command character interpretation,
- 3.) Argument processing initialization,
- 4.) Variable argument processing
- 5.) Variable argument value retrieval,

- 6.) Numeric argument processing,
- 7.) Argument value storage,
- 8.) Normal termination,
- 9.) Numerical sign processing,
- 10.) Error termination.

The six possible character classes are:

- 1.) Letter (A - Z),
- 2.) Number (0 - 9),
- 3.) Comma (,),
- 4.) End-of-string character (\$),
- 5.) Sign (+ or -),
- 6.) Other.

With the above state and character class definitions in mind, we present below the finite state transition table used by GETCOM.

CHARACTER CLASSES

	1	2	3	4	5	6
1	2	1	1	10	1	1
2	2	2	3	8	10	10
3	4	6	7	7	9	10
4	4	4	5	5	10	10
5	4	6	7	7	9	10
6	10	6	7	7	10	10
7	4	6	7	7	9	10
8	10	10	10	10	10	10
9	10	6	10	10	9	10

In order to illustrate the use of the transition table without going into too much detail, one set of state transitions will be explained here. Consider state 4, the variable argument processing state. Its transition entries are: 4, 4, 5, 5, 10, 10. The interpretation is that if the current state of the system is 4 and the next character in the instruction string is a letter, then we stay in state 4 and control in the program is transferred (by use of a COMPUTED GO TO using 4) to a section of code that builds up the variable name. If the next character is a digit, the

the same applies since a variable name can consist of any 2 characters so long as the first is a letter, which situation is guaranteed at this point by the fact that we are already in state 4. However, if the next character is either a comma or the end of the string we move to state 5 which attempts to retrieve the value of the variable from the symbol table by calling HASH. If the next character is a numeric sign, this is an error since imbedded signs are not allowed. Finally, an illegal character also sends control to the error termination state 10.

When the string processing is complete, GETCOM will have placed the hashing key associated with the command part of the instruction into the variable FCT. That key is required by MAIN. GETCOM will also have stored each one of the parameters of the instruction in PAR(2) through PAR(N + 1), where N is the number of arguments in the instruction. PAR(1) will have been set to N. In the case of variables in the instruction string, PAR will contain the variable's value that was retrieved from the symbol table. In the case of numeric arguments in the string, PAR will contain the binary (signed or unsigned) equivalent of the Hollerith number that was in the string.

HASH

FORTTRAN REFERENCE:

IVAL = HASH(KEY, FOUND)

FUNCTION:

To provide a simple, yet effective method for retrieving values from the Editor's symbol table.

ERROR CONDITIONS:

Because of the method (see below) used to index the symbol table, a KEY value of 0 is not allowed. If this occurs, an error message appears. If the symbol table becomes full, an error is also indicated.

COMMENTS:

In the case of a full symbol table error, as the error message given in section 7 indicates, the operator has two options: delete some variables from the table, or redimension. If redimensioning is chosen, note that the dimension of both KEYS and VALUES must be a power of 2 and that that power of 2 must be the value of N that is set in the DATA statement. Note also that if the dimensions are changed, they must be changed everywhere that the labeled COMMON block, BLK2, appears; viz. in MAIN, HASH, DEHASH, and INHASH.

The symbol table is indexed by use of a "hashing function" applied to the key reference (the variable name or some encodement thereof, which is the case with the Editor). For more on hashing with symbol tables, see p. 43 of [2].

HEXDMP

FORTTRAN REFERENCE:

CALL HEXDMP(INBUF,MIN,IBEG,IEND)

FUNCTION:

To provide a hexadecimal dump of (IBEG-IEND+1) words of the input tape buffer, beginning with the IBEG'th word.

ERROR CONDITIONS:

None

COMMENTS:

None

INHASH

FORTTRAN REFERENCE:

CALL INHASH(KEY,VALUE)

FUNCTION:

To install a new variable name and value in the symbol table maintained by subroutine HASH.

ERROR CONDITIONS:

An error obtains if the variable name already exists.

COMMENTS:

KEY is the same KEY defined in the documentation of HASH.
VALUE is the value that the new variable is to have.

INIT

FORTTRAN REFERENCE:

CALL INIT(IFILE,MIF1,MIF2)

FUNCTION:

To initialize the whole Data Editing system.

ERROR CONDITIONS:

Only error returns from the disk operations that are performed.

COMMENTS:

This routine initializes the I/O control blocks that are used in disk access (see [1]). It also sets up the disk files which are to be used and installs the command references in the symbol table. The command reference values are composed of two parts: the subroutine reference number and the parameter position number, both of which are packed into the value. The subroutine reference number has been discussed elsewhere (see MAIN and GETCOM). The parameter position number is the index into PAR where the day-time values of the instruction begin. For those instructions which do not have day-time references, this value is zero.

A word of explanation is in order, regarding the key values used in initializing the commands. They are obtained by taking the internal code equivalent (see CODIT), of the command characters, multiplying the first number by 36 and adding the second. For example, the first entry in the FCN table in INIT is for the RT command. The internal code equivalent of an R is 27, that of a T is 29. And $27 \times 36 + 29 = 1001$, the first of the two entries in the first column of FCN. The second entry, 0100, signals that 1 is the subroutine reference number (the COMPUTED GO TO in MAIN will cause a jump to its first argument) and that this command has no day-time references.

Future plans for this routine include options for a restart where the previous edit run left off. These options could be very valuable if it develops that one run, because of DAS recording errors, becomes much less than automatic in that the person performing the edit must manipulate the data extensively. If this turns out to be the case and the monitor must quit a run before it is complete, then without some restart capabilities, all accomplished to that point will be lost, and the monitor would have to resume from scratch later on.

LSHIFT

FORTTRAN REFERENCE:

CALL LSHIFT(IBUF,ICNT,M,N)

FUNCTION:

To perform a left shift by characters (2 to a word) of IBUF. Words shifted out of are zero filled.

ERROR CONDITIONS:

None

COMMENTS:

Characters M through N of IBUF are shifted ICNT positions.
CAUTION: be sure not to shift out of IBUF. This is a machine language routine.

MAIN

FORTTRAN REFERENCE:

None, this is the main program.

FUNCTION:

To provide complete control, through the use of subroutine calls, over the whole editing process.

ERROR CONDITIONS:

Unknown command function requested.

COMMENTS:

The MAIN routine functions by calling READIN to get an instruction string brought into core, CODIT to convert the string to internal code, GETCOM to convert those characters into a command reference key and list of parameters, HASH to retrieve the subroutine reference number and parameter position code, and finally by using the subroutine reference number in a COMPUTED TO GO to jump to a subroutine call which effects the operation desired.

New commands are entered by entering the subroutine call here, and entering the command reference in INIT.

Note that all arrays are dimensioned in MAIN and are in blank common. This was done to save storage on the RAYTHEON: the system provides that the Resident loader and blank common begin at the same location, so that by putting all arrays in common, even if they don't get used that way (in other COMMON statements in the subroutines), at least the storage allotted to the Resident Loader is recovered and rendered useable.

Note also that all the arrays that are passed to the various routines have their dimensions passed with them by means of a variable which is set to the dimension value in a DATA statement in MAIN. This allows the user to change the size of critical arrays with only two program changes in MAIN, namely the DATA statement value and the DIMENSION statement value, rather than having to change in each subroutine also.

OPENIT

FORTTRAN REFERENCE:

CALL OPENIT(IOT,L)

FUNCTION:

To link to FCS's routine OPENFL, which opens an already-created disk file for use.

ERROR CONDITIONS:

See CLOSE documentation.

COMMENTS:

See CLOSE documentation.

OUTHEX

FORTRAN REFERENCE:

CALL OUTHEX(IBUF,N,CODE,SKIP)

FUNCTION:

To output N words of IBUF in a hexadecimal format.

ERROR CONDITIONS:

None

COMMENTS:

The numbers are output in a fixed Z4 format (4 hex characters per word). See OUTINT and OUTTXT write-ups for more.

OUTINT

FORTRAN REFERENCE:

CALL OUTINT(IBUF,N,CODE,SKIP)

FUNCTION:

To output integer numbers to the list device (CRT). The numbers are stored in IBUF. N is the number of numbers to be output. CODE and SKIP are as in OUTTXT.

ERROR CONDITIONS:

None

COMMENTS:

The numbers are output in a fixed I6 format including sign, and follow the same concepts outlined under OUTTXT. See the write-up of OUTTXT for more.

OUTTXT

FORTRAN REFERENCE:

CALL OUTTXT(IBUF,N,CODE,SKIP)

FUNCTION:

To output Hollerith text to the list device (CRT). The text is stored in IBUF, two characters per word. N is the number of words to output, CODE is a carriage control code, and SKIP is the number of blanks to output before starting.

ERROR CONDITIONS:

None

COMMENTS:

This provides for stream-oriented output, and was created because space was at a premium in the RAYTHEON and the FORTRAN I/O package required on the order of 2500 words of storage. This routine (and its sister routines OUTINT and OUTHEX) together require an order of magnitude less, in addition to being much easier to use.

CODE, as was mentioned above, is a carriage control which controls the output as follows (LF = line feed, CR = carriage return):

```
CODE = 0,          do nothing and return,
CODE = 1,          output      (LF)  (text)  (CR),
CODE = 2,          output              (text)      ,
CODE = 3,          output      (LF)  (text)      ,
CODE = 4,          output              (text)  (CR),
CODE = 5,          ouptut      (LF)              (CR),
CODE = 6,          output              (CR),
CODE = 7,          output      (LF)              ,
CODE = other,      do nothing and return.
```

Note that if $N > 72$, a CR/LF is inserted before the 73rd character so that the stream may continue. This requires the keeping of an internal cursor position which is used only by OUTTXT, OUTINT, and OUTHEX and is reset to zero only when a CR is sent to the CRT for any reason (internally because $N > 72$ or externally through CODE). Thus, unless one ends with a CR, the next output command may be started in the middle of a line.

See the listings of OUTTXT, OUTINT, and OUTTXT in the appendix for more.

PDISK-FUTURE

FORTRAN REFERENCE:

IERR = PDISK(PAR,MPA,OUTBUF,MOU,CLSTAB,MCL,WRTSWT,LRECF)

FUNCTION:

To print out disk records according to the following example format:

NO.	VAL.	FLAG	NO.	VAL.	FLAG	etc.
1	19483	0001	2	15421	0011	...

ERROR CONDITIONS:

Only that the request parameters are out of range.

COMMENTS:

PAR(2) in this case is a switch, which determines what is printed as follows:

	0, print whole record;
	1, print all values;
if PAR(2) =	2, print all flags;
	3, print values and flags for
	class variable in PAR(3);
	4, print values for class variable;
	5, print flags for class variable.

POUTBF

FORTRAN REFERENCE:

CALL POUTBF(OUTBUF,MOV,IBEG,IEND)

FUNCTION:

To print out the contents of the disk buffer area in a formatted way.

ERROR CONDITIONS:

None

COMMENTS:

This subroutine prints, to the CRT, first a line containing the label and time information stored in the first 6 words of OUTBUF, and then a series of lines containing the measurement values numbered from IBEG through IEND. If IBEG is 0, i.e., the instruction PO which calls this routine has no arguments, then just the header information is printed. If IEND is less than IBEG, then just the value IBEG is printed after the header.

RDTAPE

FORTRAN REFERENCE:

CALL RDTAPE(INBUF,MIN,IERR)

FUNCTION:

To effect the reading of one record from the raw data tape.

ERROR CONDITIONS:

RDTAPE checks error returns from the READMT routine and prints error message accordingly.

COMMENTS:

This is a FORTRAN subroutine which simply sets up to call READMT: the latter does the work of reading the EBCDIC formatted tape.

READIN

FORTTRAN REFERENCE:

CALL READIN(IBUF,N)

FUNCTION:

To provide variable length input from the keyboard device. Up to 72 characters from the keyboard (usually CRT) will be read in and stored in IBUF, 2 characters per word. N will be set to the number of characters read.

ERROR CONDITIONS:

None

COMMENTS:

This routine was necessary because no variable length input was allowed from FORTRAN. If a format called for 72 characters of input, there had to be 72 characters typed in. This routine pads out the buffer with blanks if less than 72 are typed in. IBUF must be at least 36 words long.

READIN is written in machine language.

READMT

FORTTRAN REFERENCE:

CALL READMT(IBUF,ICNT,NCHAR,IERR)

FUNCTION:

To read one record into IBUF from the EBCDIC coded, 9 track, DAS generated, magnetic tape, referred to elsewhere as the input tape or raw data tape, and to perform physical, magnetic tape-type error checks.

ERROR CONDITIONS:

IERR will contain information on the number and type of mag tape errors that occurred. If IERR = 0, no errors occurred. If IERR = -1, a single error applying to the entire record occurred. If IERR = -32767, the record was completely unreadable. If IERR > 0, then IERR is the number of parity or other physical tape type errors in the record.

COMMENTS:

When the subroutine is entered, ICNT represents the number of characters to skip on the input tape record, before beginning the transfer of data into IBUF. If used, ICNT must be a positive, even integer. The routine sets ICNT to the number of characters read when it passes control back. If the sign of ICNT is negative, the record on the tape was too long and more data needs to be read. If ICNT is zero, an EOF was encountered. Also on entering, NCHAR should be set to the maximum number of characters to transfer into IBUF from the tape. NCHAR also must be a positive, even integer.

ICNT and NCHAR were designed in the above way to handle the possible occurrence of preemptive scans on the data tape. A preemptive scan is one that occurred in an alarm condition or simply a manual condition, and could begin in the middle of a good scan. Were this to happen, an unusually long "scan" would result, which could, nevertheless, be processed using this subroutine via the RT command.

For more on this, see the RT command in section 4, and the listing in the appendix.

This routine is in machine language.

RSHIFT

FORTRAN REFERENCE:

CALL RSHIFT(IBUF,ICNT,M,N)

FUNCTION:

To perform a right shift by characters (bytes) of IBUF. The opposite of LSHIFT.

ERROR CONDITIONS:

None

COMMENTS:

See LSHIFT documentation.

RWNDIT

FORTRAN REFERENCE:

CALL RWNDIT(L)

FUNCTION:

To rewind unit L.

ERROR CONDITIONS:

None

COMMENTS:

This routine is written in machine language. It was necessary to do so, rather than just using the FORTRAN REWIND instruction, because of storage problems. As the Editor is now, no FORTRAN mag tape operations are used. The whole tape operations section of the FORTRAN library is not loaded in with the Editor, saving approximately 2000 words of storage.

SEEKEF

FORTRAN REFERENCE:

CALL SEEKEF(N,IREC,L)

FUNCTION:

To find N ends-of-file on unit L.

ERROR CONDITIONS:

None

COMMENTS:

IREC will be set to the number of records passed over. This includes the EOF's passed over. This is a machine language routine.

SKIP

FORTRAN REFERENCE:

CALL SKIP(N,L)

FUNCTION:

To skip N records forward on unit L.

ERROR CONDITIONS:

End-of-tape marker reached before completing the command.

COMMENTS:

If $N < 0$ on entering the subroutine, then this subroutine functions like the BACKSP subroutine. On leaving the subroutine, $|N|$ will be the number of records passed over. If $N < 0$ here, the EOF was reached. This is a machine language routine. For more on this routine see the SK command in section 4, and the listing in the appendix.

SPLITA

FORTTRAN REFERENCE:

CALL SPLITA(ARG1,ARG2,ARG3)

FUNCTION:

To separate out the bytes of ARG1 and right justify them respectively in ARG2 and ARG3.

ERROR CONDITIONS:

None

COMMENTS:

None

STATS-FUTURE

FORTTRAN REFERENCE:

IERR = STATS(PAR,MPA,TABLE,MTA1,MTA2,OUTBUF,MOU,CLSTAB,MCL)

FUNCTION:

To produce a set of statistics on the portion of the disk-stored data defined in the PAR vector by the class variable and day-time positions, that should give some indication of whether all is well with the data collection mechanisms.

ERROR CONDITIONS:

None

COMMENTS:

The statistics to be gathered are all relative ones, which means among other things that they are dimensionless quantities whose values will all be between ± 2 . If any value is reported larger in either direction, that is an indication that some measuring device recorded values larger than expected. (One would then zero in on the data to determine which device malfunctioned during what time interval.)

In order to implement this routine, certain types of stored statistics on the measured values are required. There are, for each measurement variable:

- 1) $\min_p$ = expected minimum value,
- 2) $\max_p$ = expected maximum value,
- 3) y_p = typical value,
- 4) Δy = expected change in y over time,
- 5) $\Delta y/y$ = expected relative deviation.

The standardized statistics to be reported are:

- 1.) standardized min:
$$(\min - \min_p) / (\max - \min_p),$$
- 2.) standardized max:
$$(\max - \min_p) / (\max_p - \min_p),$$
- 3.) standardized value:
$$(y - \min_p) / (\max_p - \min_p),$$
- 4.) standardized standard deviation (SD):
$$SD / SD_p,$$
- 5.) standardized relative standard deviation (RELSD)
$$RELSD / RELSD_p.$$

These statistics could all be computed in one pass through the data values, and would be easy to implement. There are more like this that could be useful in providing early detection of measurement device anomalies, but these would be developed and implemented on an as-needed basis.

It should be noted that these statistical checks are simple, gross ones designed to provide the simplest of statistical data checks, and should not in any way be associated with providing information on data reliability.

STDATA

FORTRAN REFERENCE:

IERR = STDATA(PAR,MPA,OUTBUF,MOU,CLSTAB,MCL,WRTSWT,LRECF)

FUNCTION:

To provide the ability to set data or flag values over a range of measurement variables and time definitions of the disk-stored data scans.

ERROR CONDITIONS:

Since one of the parameters in PAR is a class variable or its degenerate form, a measurement variable number, and since STDATA must interpret this reference, an illegal variable reference condition could obtain.

COMMENTS:

This routine is the one eventually got to by typing in an SD instruction to the Editor. After DRIVER brings in one of the records referenced, STDATA interprets the class variable and sets the data or flag values as required. STDATA sets the write data switch on, which puts DRIVER in the read and write mode.

TRNRAW

FORTRAN REFERENCE:

CALL TRNRAW(INBUF,MIN,OUTBUF,MOU, TABLE,MTA1,MTA2,NREC,KPRNT)

FUNCTION:

To provide for the automatic transfer of data from the raw data tape to core and then to disk.

ERROR CONDITIONS:

None that are endemic to TRNRAW. The errors that can occur, do so in the subroutines that TRNRAW calls.

COMMENTS:

This is the routine that one calls by giving the editor a TR command. It is a level 3 routine (see section 2), and loops through RDTAPE, CONALL, UTABLE and DSKRTS for NREC records. It produces monitoring prints after every KPRNT records are transferred.

If an EOF is encountered or any error condition occurs in one of the routines called by this one, processing terminates and a message indicating the exact number of records transferred is printed.

UNSPLT

FORTRAN REFERENCE:

CALL UNSPLT(ARG1,ARG2,ARG3)

FUNCTION:

To reverse the process of subroutine SPLITA. That is, where SPLITA unpacks bytes into the second and third arguments, UNSPLT packs the right bytes of ARG2 and ARG3 into ARG1.

ERROR CONDITIONS:

None

COMMENTS:

None

UTABLE

FORTRAN REFERENCE:

CALL UTABLE(OUTBUF,MOU, TABLE,MTA1,MAT2,IERR)

FUNCTION:

To update the table of contiguous times (TABLE) with the time of the record about to be written onto the disk, which record currently resides in the disk buffer, OUTBUF.

ERROR CONDITIONS:

If the number of entries is about to exceed the dimensions of TABLE, which are MTA1 and MTA2, then no entry is made, and an error message is printed. Also, if an entry is attempted for a record whose time is out of sequence, that entry is not made and an error message appears.

COMMENTS:

The table of contiguous times is used as an index table into the disk-stored data scans. The idea is to keep a record of contiguous blocks of data scans whose time differences, Δt , from one record to the next are all the same. Given a beginning time for such a group, a record number associated with that beginning time, the Δt , and the time of a scan one is interested in, the record number of the scan of interest can be quickly calculated. This of course means that every occurrence of a time difference between consecutive records, that differs from that between the 2 previous records, causes an entry in the TABLE.

The data for the time table is stored by columns so that (TABLE(J,NENT), J=1,6) represent in order: the day of the first entry for this contiguous group, the time in minutes for the first entry in the group, the record number associated with that entry, the day of the last entry in the group, the time in minutes of the last entry, and finally the time difference, Δt , that applies for that group.

WRITEF

FORTRAN REFERENCE:

CALL WRITEF(N,L)

FUNCTION:

To write 3 EOF's on unit L.

ERROR CONDITIONS:

None

COMMENTS:

See write-up of RWNDIT. Comments there apply here also.

WRITMT

FORTRAN REFERENCE:

CALL WRITMT(IBUF,N,M,IERR,LUN)

FUNCTION:

To write the contents of the IBUF out to the 1108 compatible tape mounted on logical unit LUN. N is the number of items

(words or characters depending on the value of M) to be transferred, and M is a format control code.

ERROR CONDITIONS:

IERR will be set <0 if an unrecoverable mag tape error occurred. Otherwise IERR=0.

COMMENTS:

This is a somewhat complicated routine, that was designed to provide capabilities other than just those required by the Total Energy Data Editor. The primary requirement for the Data Editor regarding this routine was to write out the 16 bit words in IBUF in such a way that the records could be read on NBS's UNIVAC 1108 with a minimal amount of work at both ends (RAYTHEON and 1108).

For a detailed explanation of what WRITMT does with the various values of N and M, see the subroutine listing in the appendix. Suffice it to say, here, that N is the number of words to write out, and M should be equal to -1. These comments are true so long as the output tape is mounted on the 7 track drive (physical unit 14).

7. THE DATA EDITOR - HOW IT TALKS TO YOU

Any Data Editor error message is composed of three parts: the name of the subroutine that was in control when the error occurred, the error type number, and a list of arguments. This appendix provides explanations of the various error types. The number reported in the displayed error message is the one used to determine which message to read below.

* * *

1. Unknown command function requested. Check command requested, and retype the instruction.
2. Illegal character in instruction string. ARG1 is the position in the string of this character.
3. A variable argument in the instruction has not been defined previously, and therefore cannot be used as an argument. ARG1 is the number of the argument in error.
4. Incorrect instruction format. Embedded letters in numbers, or embedded signs, are examples.
5. Symbol table hash key equal to zero. This is a serious error. An attempt was made to retrieve a value from the symbol table with a reference index of zero. Call a programmer.
6. Symbol table is full. No more variables can be defined until space is obtained. Either delete some variables, or stop and redimension the table.
7. Failure in the INHASH routine. ARG1 is the value of FOUND. If ARG1 is 1, this simply means an attempt was made to define a variable that already is defined. Delete the original definition and then insert the new one. If ARG1 is 0, a serious system error occurred. Call a programmer.
8. Failure in the DEHASH routine, ARG1 is the value of FOUND. If ARG1 is 0, an attempt to delete an unknown variable from the symbol table was made. If ARG1 is 1, a serious system error occurred. Call a programmer.
9. An error occurred in reading the raw data tape. Either the character count (1st argument) is bad, or a magnetic tape error (2nd argument) occurred.

-32767, if record was unreadable;
-1, if single parity error occurred applying to
entire record;
ARG2 = 0, if no mag tape error occurred; or
n, where n is the number of parity errors that
occurred.

If ARG2 = 0, the error was with the character count. If ARG1 < 0, the tape record was longer than expected and some data was therefore unread. If ARG1 = 0, an end-of-file was encountered.

10. An illegal character was encountered while converting from EBCDIC to binary the measurement variable group of characters that is physically the ARG1'th such group in the scan whose day is ARG2 and whose time is ARG3. ARG5 is the hexadecimal dump of the flag word as explained in the documentation of CONGRP, which contains pointers to the position of the illegal characters.
11. Illegal EBCDIC characters were encountered while converting the label and day-time characters of the scan. ARG1 is the number of such errors.
12. The table of contiguous times (TABLE), used as an index into the disk, is full. Either redimension or stop transferring data to the disk, continue on with an edit of the data already transferred, and then start a new edit from the point left off.
13. A scan request was made for a scan whose time was not found in the table of contiguous times. Try again.
14. Disk operation error. ARG1 is the status word returned from FCS (see [1], p.39).

 0, if operation was other than below;
ARG2 = 1, if operation was a read;
 2, if a write;
 3, if an update.
15. Error return from COMINT. ARG1 = 99, if resultant number was greater than 32768, or n if the nth character was other than a digit, a +, a -, or a blank. ARG2 = n if this call was the n'th grouping call from CONALL. ARG4 is the 16 least significant bits of the value.
16. Illegal measurement variable reference in the instruction. It must be a class variable or a number between 1 and NVAR.
17. Out of space in CLSTAB, the class variable storage area. ARG1 is the amount needed for the latest request. ARG2 is the amount of space left in the table. Either delete a class variable to make room, or go back and redimension.
18. The class variable referenced was not found.
19. An expansion of a class variable definition was attempted when the class variable referenced was not physically the last one in the list. Delete the current definition and start over.

20. The input records are out of sequence. The last record whose transfer was attempted had a day-time value less than the latest one on the disk. ARG1 and ARG2 are respectively the day and time of the last record on the disk. Note that the time printed is in minutes, not hours and minutes.
21. Error return from WRITMT routine. If ARG1 = 1, the error occurred while writing the first record of an output block. If ARG1 = 2, the error occurred on the second (the data) record. ARG2 is the total number of scans that are on the disk. ARG2 and ARG3 are, respectively the day and time of the record being written when the error occurred. This error is an unrecoverable one. It is necessary to mount a new tape.

8. BIBLIOGRAPHY

1. Fetzner, M., "File Control System (FCS) Specification", RAYTHEON Report number DN 59004CWL, Revision B, November, 1973.
2. Morris, R., "Scatter Storage Techniques", Comm. of ACM, vol. II, no. 1, Jan. 1968, p.43.
3. Filliben, J.J., Jackson, R.H.F., Kirsch, R.A., Lozier, D.W., Orser, D.J., "Progress Report on Applied Mathematics Division contribution to MIUS-Jersey City Total Energy Project, June 1975.

APPENDIX A



APPENDIX A - SUBROUTINE LISTINGS

This appendix contains a listing of each of the subroutines that comprise the Total Energy Data Editor as of the freeze date for documentation of January 31, 1975. Each listing begins a new page and all are in alphabetical order with a few exceptions. The exceptions are due to the fact that although all subroutines discussed in Section 6 are conceptually unique, some of the different machine language routines (e.g., BACKSP, SKIP, SEEKEF, RWNDIT, and WRITEF) were combined into one subroutine with different entry points, in order to save storage. An index into the listings is given below.

* * * * *

<u>SUBROUTINE NAME</u>	<u>PAGE NO.</u>
ADRESS	A-1
BACKSP	A-3
CLASS	A-11
CLOSE	A-13
CODIT	A-66
COMINT	A-16
CONALL	A-19
CONATE	A-20
CONETA	A-22
CONGRP	A-25
CREATE	A-13
CVPROC	A-32
DATTAP	A-34
DECDIT	A-66
DEHASH	A-35
DELETF	A-13
DRIVER	A-36
DSKINT	A-13
DSKRD	A-13
DSKRTS	A-38
DSKUPD	A-13

DSKWT	A-13
ERRPRT	A-39
EX	A-40
GETCOM	A-41
HASH	A-43
HEXDMP	A-44
INHASH	A-45
INIT	A-46
LSHIFT	A-47
MAIN	A-52
OPENIT	A-13
OUTHEX	A-54
OUTINT	A-54
OUTTXT	A-54
POUTBF	A-64
RDTAPE	A-65
READIN	A-66
READMT	A-73
RSHIFT	A-47
RWNDIT	A-3
SEEKEF	A-3
SKIP	A-3
SPLITA	A-78
STDATA	A-80
TRNRAW	A-81
UNSPLT	A-78
UTABLE	A-82
WRITEF	A-3
WRITMT	A-83

SUBROUTINE ADDRESS

```
1 *      RETRIEVE ABSOLUTE ADDRESS OF A VARIABLE
2 *
3 *      MAY 13, 1974
4 *
5 *      CALL FROM FORTRAN4 VIA:
6 *      CALL ADDRESS (VAR,IADR)
7 *      WHERE IADR WILL BE SET EQUAL TO THE ADDRESS
8 *      OF VARIABLE VAR.
9 *
10 *     NOTE: VAR MUST BE DEFINED WITHIN FORTRAN4
11 *
12         LIBR  ADDRESS
13         NTRY  ADDRESS
14 *
15 ADDRESS  DATA  POOL
0000 0007 16      LDW   VAR      GET VARIABLE ADDRESS
0001 800A 17      LDX   IADR     GET IADR ADDRESS
0002 900B 18      STW * 0        SET IADR
0003 7800 19      SMB   R.RET
* 0004 07FF 20      JSX   R.RET   RETURN
* 0005 2004 21      DATA  POOL
0006 0007 22 *
0007 0004 23 POOL  DATA  4,0,0
0008 0000
0009 0000
000A 0000 24 VAR   DATA  0
000B 0000 25 IADR  DATA  0
26      END
```

0005 R.RET

NO ERRORS

ADDRESS	0000	IADR	000B	POOL	0007	R-RET	0005
VAR	000A						
PAS?							

SUBROUTINES BACKSP, SKIP, SEEKEF, RWNDIT, AND WRITEF

```
1 *      MAG TAPE MISCELLANEOUS OPERATIONS
2 *
3 *      SEPT 10, 1975
4 *
5 *
6 *      ALL CALLS:
7 *      SMB   BACKSPML
8 *      JSX   BACKSPML
9 *      DATA LOC OF N
10 *     DATA LOC OF LUN
11 *     RETURN WILL BE HERE
12 *     AND
13 *     CALL BACKSP(N,LUN)
14 *     CALL SKIP(N,LUN)
15 *     CALL SEEKEF(N,IREC,LUN)
16 *     CALL RWNDIT(LUN)
17 *     CALL WRITEF(N,LUN)
18 *
19 *
20 * FIRST PROGRAM:
21 *     BACKSPACE OR SKIP RECORDS
22 *     CALL FROM FORTRAN4:
23 *     CALL BACKSP(N,LUN)
24 *     OR
25 *     CALL SKIP(N,LUN)
26 *
27 *     WILL BACKSPACE UNIT "LUN" N RECORDS
28 *     OR WILL SKIP N RECORDS FOREWARD.
29 *
30 * NOTE: A MINUS N WITH BACKSP IS EQUIVALENT TO
31 *       CALLING SKIP.
32 *       A MINUS N WITH SKIP IS EQUIVALENT TO
33 *       CALLING BACKSP.
34 *
```

```

35 *      N WILL BE SET TO A NEGATIVE NUMBER IF THE
36 *      END-OF-TAPE WAS REACHED ON SKIP OR IF
37 *      THE LOAD POINT WAS REACHED ON BACKSP.
38 *      THE ABSOLUTE VALUE OF THIS NEW N WILL
39 *      BE THE ACTUAL NUMBER SKIPPED OR BACKSP.
40 *
41 *      SECOND PROGRAM:
42 *      SEEK END-OF-FILE(S)
43 *      CALL FROM FORTRAN4:
44 *      CALL SEEKEF(N,IREC,LUN)
45 *
46 *      WILL FIND "N" END-OF-FILES ON UNIT "LUN"
47 *      AND SET IREC TO THE NUMBER OF RECORDS
48 *      SKIPPED (INCLUDING THE END-OF-FILES).
49 *
50 *      THIRD PROGRAM:
51 *      REWIND
52 *      CALL FROM FORTRAN4:
53 *      CALL RWNDIT(LUN)
54 *
55 *      WILL REWIND UNIT "LUN"
56 *
57 *      FOURTH PROGRAM:
58 *      WRITE END-OF-FILES
59 *      CALL FROM FORTRAN4:
60 *      CALL WRITEF(N,LUN)
61 *
62 *      WILL WRITE N END-OF-FILES ON LUN
63 *
64 *      LIBR  BACKSPML
65 *      LIBR  BACKSP,RWNDIT,SEEKEF,SKIP,WRITEF
66 *      NTRY  BACKSPML
67 *      NTRY  BACKSP,RWNDIT,SEEKEF,SKIP,WRITEF
68 *
69 *      OPEN      EQU      66

```

70	STRT	PROC		
71		DATA	POOL	
72		LDW	P(1)	
73		JSX	SETUP	
74		ENDP		
75	*			
76	BACKSPML	STX	LUN	SAVE RETURN
77		LDW	* 0	
78		STW	N	SET N
79		LDW	BMLRET1	
80		STW	RETNML	SET NEW RETURN
81		LDW	* 1	GET LUN
82		JMP	BMLENT	
83	BMLRET	LDW	BMLRET2	
84		STW	RETNML	RESET RETURN
85		LDX	LUN	
86		JMP	* 2	RETURN
87	BMLRET1	JMP	BMLRET	
88	BMLRET2	SMB	R.RET	
89	*			
90	BACKSP	STRT	IREC	DATA DATA
91	BMLENT	EQU	BACKSP+2	LDW IREC
92		JMP	RETURN	JSX SETUP
93		JMP	SK1	
94	B1	LDW	UNIT	
95	BACK	DOT	0.0	
96		JSX	WAIT	
97		SLL	2	
98		SAP		
99		JMP	B2	
100		JSX	CHEK	
101		JMP	B1	
102		JMP	RETURN	
103	B2	LDW	CNTDOWN	
104		ADD	N1	

105		JMP	SKRET
106	*		
107	SKIP	STRT	IREF
108		JMP	RETURN
109		JMP	B1
110	SK1	LDW	UNIT
111	READ1	DOT	0,0
112		JSX	WAIT
113		CAX	
114		LDW	UNIT
115		CLB	0
116		SNE	
117		JMP	DEV0
118		CXA	
119		SLL	3
120		SAP	
121		JMP	SK2
122	DEV0	JSX	CHEK
123		JMP	SK1
124		JMP	RETURN
125	SK2	LDW	CNTDOWN
126	SKRET	LDX	N
127		JMP	SERET
128	*		
129	SEEKEF	STRT	LUN
130		JMP	SE2
131		JMP	SE2
132	SE1	LDW	UNIT
133	READ2	DOT	0,0
134		JSX	WAIT
135		SPL	4
136		SAD	
137		JMP	SE1
138		JSX	CHEK
139		JMP	SE1

140	SE2	LDX	IREC	
141		LDW	CNTUP	
142	SERET	STW *	0	
143		JMP	RETURN	
144	*			
145	RWNOIT	STRT	N	
146		NOP		
147		NOP		
148		LDW	UNIT	
149	RWNO	DOT	0,0	
150		JMP	RETURN	
151	*			
152	WRITEF	STRT	IREC	
153		JMP	RETURN	
154		JMP	RETURN	
155	W1	LDW	UNIT	
156	WEOF	DOT	0,0	
157		JSX	WAIT	
158		JSX	CHEK	
159		JMP	W1	
160	RETURN	LDW	UNIT	
161	DISCO1	DOT	0,0	
162	RETNML	SMB	R.RET	
163		JSX	R.RET	
164		DATA	POOL	
165	*			
166	SETUP	STX	RET	
167		CAX		
168		LDW *	0	GET LUN
169		STW	LUNSV	
170		OPEN		
171		DATA	FIOT	
172	H0	DATA	0,N1	
173	LUNSV	DATA	0	
174	H0	DATA	9	

175	N1	DATA	1
176		STW	UNIT
177		STB	STAT1+1
178		STB	STAT2+1
179		AND	X00F0
180		STB	DISC01+1
181		STB	DISC02+1
182		ORI	N2
183		STB	WEOF+1
184		ORI	N8
185		STB	RWND+1
186		ORI	N1
187		STB	BACK+1
188		AND	X00F9
189		STB	READ1+1
190		STB	READ2+1
191		LDW	UNIT
192	DISC02	DOT	0.0
193		CLR	
194		STW	CNTDOWN
195		STW	CNTUP
196		LDX	N
197		LDW	* 0
198		LDX	RET
199		SAX	
200		JMP	\$+2
201		JMP	ISZERO
202		SAP	
203		IXS	1
204		IXS	2
205		NOP	
206		SAP	
207		CMP	
208	ISZERO	STW	NCNT
209		JMP	* 0

210	*		
211	WAIT	LDW	UNIT
212	STAT1	DIN	0.0
213		SRC	L 2
214		SAM	
215		JMP	WAIT
216		LDW	CNTDOWN
217		SUB	N1
218		STW	CNTDOWN
219		LDW	CNTUP
220		ADD	N1
221		STW	CNTUP
222		LDW	UNIT
223	STAT2	DIN	0.0
224		JMP	* 0
225	*		
226	CHEK	LDW	NCNT
227		SUB	N1
228		STW	NCNT
229		CMW	N0
230		SLE	
231		JMP	* 0
232		JMP	* 1
233	*		
234	N2	DATA	2
235	N8	DATA	8
236	X00F0	DATA	X'00F0'
237	X00F9	DATA	X'00F9'
238	FET	DATA	0
239	*		
240	F10T	DATA	0
241	UNIT	DATA	0
242	CNTDOWN	DATA	0
243	CNTUP	DATA	0
244	NCNT	DATA	0

245 *
246 POOL DATA 5,0,0
247 N DATA 0
248 IREC DATA 0
249 LUN DATA 0
250 *
251 END
BE

SUBROUTINE CLASS

```
1 *      INTEGER FUNCTION CLASS(K)
2 *
3 *      DECEMBER 17, 1974
4 *
5 *      CLASS WILL BE SET AS FOLLOWS:
6 *          K<10  2
7 *          K<36  1
8 *          K=36  3
9 *          K=41  4
10 *         K=38 OR K=39  5
11 *         ALL OTHERS  6
12 *
13 *      LIBR  CLASS
14 *      NTRY  CLASS
15 *
16 CLASS    DATA  0
17          STW   SAVE
18          CAX
19          IXS   3
20 SAVE     DATA  0
21          LDX  * 0
22          SXP
23          JMP   $-2
24          LDW  * 0      GET K
25          SAF
26          JMP   SET6
27          CMW   N41
28          SLE
29          JMP   SET6
30          SNE
31          JMP   SET4
32          CLB   9
33          SGR
34          JMP   SET2
```

35	CLB	35
36	SGR	
37	JMP	SET1
38	CLB	36
39	SNE	
40	JMP	SET3
41	CLB	38
42	SEQ	
43	CLB	39
44	SNE	
45	JMP	SET5
46 SET6	LLB	6
47	JMP	SETIT
48 SET5	LLB	5
49	JMP	SETIT
50 SET4	LLB	4
51	JMP	SETIT
52 SET3	LLB	3
53	JMP	SETIT
54 SET2	LLB	2
55	JMP	SETIT
56 SET1	LLB	1
57 SETIT	SLL L	8
58	LDX	SAVE
59	IXS	2
60	NOP	
61	LDX *	0
62	SXP	
63	JMP	\$-2
64	STW *	0
65	LDX	SAVE
66	UNM	
67	IXS	4
68 N41	DATA	41
69	SMB	R.EXEC
70	JMP	R.EXEC
71	END	

SUBROUTINES CLOSE, CREATE, DELET, DSKINT, DSKRD, DSKUPD, DSKWT, OPENIT

1 * FCS ACCESS AND INTERFACE SUBROUTINE

2 *
3 * FEBRUARY 24, 1975

4 *
5 * FORTRAN4 ENTRY ACCESS:

6 * CALL NAME (IOT,L)

7 * WHERE:

8 * NAME CAN BE ANY OF THE FOLLOWING AND
9 * THEIR FUNCTION WILL CORRESPOND TO THE
10 * FUNCTIONS DESCRIBED IN THE RAYTHEON FCS
11 * MANUAL FOR THE "=" NAME.

12 * CLOSE = CLOSEFL
13 * CREATE = CREATEFL
14 * DELET = DELETF
15 * DSKINT = INITLIZE
16 * DSKRD = GETREC
17 * DSKUPD = UPDATE
18 * DSKWT = PUTREC
19 * OPENIT = OPENFL

20 * IOT IS THE LOCATION OF THE FCS I/O TABLE
21 * DESCRIBED IN THE RAYTHEON FCS MANUAL.

22 * L IS A CODED WORD WHICH WILL BE SET BY THIS
23 * ROUTINE TO THE FOLLOWING:

24 * L<0 . OPERATION COMPLETE, ERROR
25 * ENCOUNTERED, THE ABS. VALUE
26 * OF L WILL EQUAL THE ERROR
27 * CODE DESCRIBED IN THE RAY-
28 * THEON FCS MANUAL.
29 * L>-1 . OPERATION COMPLETE, NO ERROR.
30 * L=# OF WORDS TRANSFERRED.

31 *
32 * LIBR CLOSE,CREATE,DELET,DSKINT,DSKRD
33 * LIBR DSKUPD,DSKWT,OPENIT,LOCKIT,PLACIT
34 * NTRY CLOSE,CREATE,DELET,DSKINT,DSKRD

35		NTRY	DSKUPD,DSKWT,OPENIT,LOCKIT,PLACIT
36	*		
37	SETT	PROC	
38		DATA	POOL
39		JSX	BEGIN
40		DATA	P(1)
41		ENDP	
42	*		
43	CLOSE	SETT	CLOSEFL
44	CREATE	SETT	CREATEFL
45	DELETF	SETT	DELETEFL
46	DSKINT	SETT	INITLIZE
47	DSKRD	SETT	GETREC
48	DSKUPD	SETT	UPDATE
49	DSKWT	SETT	PUTREC
50	OPENIT	SETT	OPENFL
51	LOCKIT	SETT	LOCKFL
52	PLACIT	SETT	INSERT
53	*		
54	BEGIN	STX	RET SAVE REFERENCE
55		LDX	IOT
56		STX	IOTP SET IOT POINTER
57		CLR	
58		STW * 4	SET NO ERR ACTION
59		STW * 5	SET NO END ACTION
60		LDX	RET GET REFERENCE
61		LDX * 0	GET TRANSFER
62		JSX * 0	TRANSFER TO FCS
63	IOTP	DATA	0 I/O TABLE POINTER
64		JMP	ERROR RETURN IF ERROR
65		CLR	RETURN IF NORMAL
66	ERROR	CMP	FORCE (-) IF ERROR
67		LDX	L
68		STW * 0	SET L
69		SMB	R.RET

```
70 JSX R RET
71 DATA POOL
72 *
73 RET DATA 0
74 *
75 POOL DATA 4.0.0
76 IOT DATA 0
77 L DATA 0
78 *
79 END
BE
```

SUBROUTINE COMINT

```

1 *      COMINT - TOTAL ENERGY SUBROUTINE
2 *
3 *      DECEMBER 18, 1974
4 *
5 *      CALLING SEQUENCE:
6 *      CALL COMINT( IBUF, N, IERR, IVAR )
7 *
8 *      WILL CONVERT N CODED CHARACTERS STARTING
9 *      WITH IBUF INTO A BINARY NUMBER AND
10 *     WILL SET IVAR TO SAID NUMBER.
11 *
12 *     IERR WILL BE SET TO ZERO IF THE RANGE
13 *     OF IVAR IS WITHIN -32767 TO +32767,
14 *     OTHERWISE, IERR WILL BE SET TO 99 OR
15 *     TO THE OFFENDING CHARACTER NUMBER IF
16 *     THE CHAR IS NOT A SP, -, +, OR NUMBER.
17 *
18 *     LIBR  COMINT
19 *     NTRY  COMINT
20 *
21 RETURN  SMB  R.RET
22        JSX  R.RET
23 COMINT  DATA POOL
24        CLR
25        STW  SIGN      SET SIGN +
26        STW  NUM       INITIALIZE NUM
27 CHEKTHRU STB  LDWI+1  SET FIRST CHAR
28        LDX  N
29        CMW  * 0
30        SLS
31        JMP  THRU      ARE WE THRU?
32        LDX  IBUF      YES
33 LDWI    LDW  * 0      NO
34        SAP           GET CHARACTER

```

35		JMP	ERROR	
36		CLB	9	
37		SGR		
38		JMP	MULTADD	NUMBER
39		CLB	37	
40		SNE		
41		JMP	CONTIN	SPACE
42		CLB	38	
43		SNE		
44		JMP	CONTIN	PLUS
45		CLB	39	
46		SNE		
47		JMP	MINUS	MINUS
48	ERROR	CLR		
49		LDB	LOWI+1	
50		ADD	N1	CALCULATE OFFENDER
51	FINISH	LDX	IERR	
52		STW	* 0	SET IERR
53		LDW	NUM	
54		LDX	SIGN	
55		SXP		
56		CMP		FORCE PARITY
57		LDX	IVAR	
58		STW	* 0	SET IVAR
59		JMP	RETURN	
60	*			
61	MINUS	LDW	\$	
62		STW	SIGN	SET SIGN -?
63		JMP	CONTIN	
64	*			
65	THRU	LDX	NUM	
66	ERROR3	CLR		
67		SXP		WAS THERE OVERFLOW?
68	ERROR2	LLB	99	YES
69		JMP	FINISH	NO

```

70 *
71 MULTADD STW TEMP SAVE NUMBER
72 LLE 10
73 MPY NUM
74 ADD TEMP
75 STW TEMP
76 CLR
77 SNO
78 JMP ERROR2
79 DXS 1
80 JMP ERROR2
81 LDW TEMP
82 SAP
83 JMP ERROR3
84 STW NUM
85 CONTIN CLR JUST CONTINUE ON NOW
86 LDB LDWI+1
87 ADD N1 SET UP FOR NEXT NUM
88 JMP CHEKTHRU
89 *
90 SIGN DATA 0
91 NUM DATA 0
92 TEMP DATA 0
93 N1 DATA 1
94 *
95 POOL DATA 6,0,0
96 IBUF DATA 0
97 N DATA 0
98 IERR DATA 0
99 IUAR DATA 0
100 *
101 END
BE

```

```

S      LIBR  CONALL
      SUBROUTINE CONALL(INBUF,MIN,OUTBUF,MOU,IERR)
      INTEGER INBUF,IGRP,OUTBUF,DEBUG
      DIMENSION INBUF(MIN),OUTBUF(MOU),IGRP(5)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      INC=MOU/2-3
      IERR=0
C      CONVERT HEADER TO INDIVIDUAL BINARY VALUES.
      CALL CONETA(INBUF,OUTBUF,19,IFLAG)
      IERR=IERR+IFLAG
      IF (IFLAG.GT.0) CALL ERRPRT(11,11,IFLAG,0,0,0,0)
      CALL CODIT(OUTBUF,OUTBUF,19)
C      COMBINE FIRST 12 CHARS INTO 3 BINARY VALUES.
      DO 5 I=1,3
      CALL COMINT(OUTBUF((I-1)*4+1),4,IFLAG,OUTBUF(I))
      IERR=IERR+IFLAG
      IF (IFLAG.GT.0) CALL ERRPRT(11,15,IFLAG,1,0,0,OUTBUF(I))
5      CONTINUE
C      BUILD UP AND STORE THE DAY.
      CALL COMINT(OUTBUF(13),3,IFLAG,OUTBUF(4))
      IERR=IERR+IFLAG
      IF (IFLAG.GT.0) CALL ERRPRT(11,15,IFLAG,4,0,0,OUTBUF(4))
C      BUILD UP AND STORE THE TIME.
      CALL COMINT(OUTBUF(16),4,IFLAG,OUTBUF(5))
      IERR=IERR+IFLAG
      IF (IFLAG.GT.0) CALL ERRPRT(11,15,IFLAG,5,0,0,OUTBUF(5))
C      CONVERT, BUILD UP, AND STORE EACH OF THE MEASUREMENT GROUPS.
      DO 10 I=1,INC
      OUTBUF(I+6)=0
10     OUTBUF(I+6+INC)=1
      N=7
      DO 20 I=1,NVAR
      N=N+13
      CALL CONGRP(INBUF,IGRP,N,IFLAG)
      IERR=IERR+IFLAG
      IF (IGRP(2).GT.0) GO TO 20
      ICHAN=IGRP(1)
      IF (IFLAG.EQ.0) GO TO 15
      CALL ERRPRT(11,10,1,OUTBUF(4),OUTBUF(5),0,IFLAG)
      IF (DEBUG.EQ.11) CALL OUTINT(IGRP,5,1,0)
15     IF (ICHAN.LT.10.OR,ICHAN.GT.180) GO TO 20
      OUTBUF(ICHAN+6)=IGRP(4)
      OUTBUF(ICHAN+6+INC)=0
20     CONTINUE
      RETURN
      END

```

C;

SUBROUTINE CONATE

```
1 *      CONVERT ASCII TO EBCD
2 *
3 *      FEBRUARY 24, 1975
4 *
5 *      CALL FROM FORTRAN4 VIA:
6 *      CALL CONATE(IBUF1,IBUF2,ICNT,IERR)
7 *
8 *      WHERE: ICNT ASCII CHARACTERS FROM IBUF1 WILL BE
9 *              CONVERTED TO EBCD AND PLACED IN IBUF2.
10 *      IBUF1 & IBUF2 MAY OR MAY NOT BE SAME.
11 *      IERR WILL BE SET TO NUM ILLEGAL CHAR.,
12 *      EACH OF WHICH WILL BE SET TO ZERO.
13 *
14 *      LIBR  CONATE
15 *      NTRY  CONATE
16 *
17 CONATE  DATA  POOL
18          LDW   IBUF2
19          SLL   1
20          STW   NXST      SET FIRST STORE
21          LDW   IBUF1
22          SLL   1
23          STW   NXLD      SET FIRST LOAD
24          LDX   ICNT
25          LDW   * 0
26          STW   COUNT     SET COUNT
27          CLR
28          STW   ERR        SET NO ERRORS
29 P01     LDX   COUNT
30          OXS   1          THRU?
31          JMP   P02        NO
32          LDX   IERR       YES
33          LDW   ERR
34          STW   * 0        SET ERROR COUNT
```

35		SMB	R RET	
36		JSX	R RET	RETURN TO FORTRAN4
37		DATA	POOL	
38	P02	STX	COUNT	
39		LDX	NXLD	
40		LDL	* 0	
41		IXS	1	
42		NOP		JUST TO BE SAFE
43		STX	NXLD	
44		SMB	CC1TEBCD	
45		JSX	CC1TEBCD	
46		LDX	NXST	
47		STL	* 0	
48		IXS	1	
49		NOP		JUST TO BE SAFE
50		STX	NXST	
51		LDX	ERR	
52		IXS	1	
53	NXLD	DATA	0	
54		CLB	0	
55		SNE		ERROR?
56		STX	ERR	YES
57		JMP	P01	NO
58	*			
59	COUNT	DATA	0	
60	ERR	DATA	0	
61	NXST	DATA	0	
62	*			
63	POOL	DATA	6.0.0	
64	IBUF1	DATA	0	
65	IBUF2	DATA	0	
66	ICNT	DATA	0	
67	IERR	DATA	0	
68		END		

BE

SUBROUTINE CONETA

```
1 *      CONVERT EBCD TO ASCII
2 *
3 *      FEBRUARY 24, 1975
4 *
5 *      SOURCE ON 9 TRACK TAPE # 5
6 *
7 *      CALL FROM FORTRAN4 VIA:
8 *      CALL CONETA( IBUF1, IBUF2, ICNT, IERR )
9 *
10 * WHERE: ICNT EBCD CHARACTERS FROM IBUF1 WILL BE
11 *         CONVERTED TO ASCII AND PLACED IN IBUF2.
12 *         IBUF1 AND IBUF2 MAY OR MAY NOT BE THE
13 *         SAME.
14 *         IERR WILL BE SET TO THE NUMBER OF ILLEGAL
15 *         CHARACTERS (EACH OF WHICH WILL BE SET TO
16 *         AN ASCII "*", IE: X'AA' OR NUMBER 170).
17 *
18 * NOTE: IBUF1, IBUF2, ICNT, AND IERR MUST BE SINGLE
19 *        INTEGER LABEL COMMON DATA.
20 *
21 *      LIBR  CONETA
22 *      NTRY  CONETA
23 *
24 CONETA  DATA  POOL
25          LDW   IBUF2
26          SLL   1
27          STW   NXST      SET FIRST STORE
28          LDW   IBUF1
29          SLL   1
30          STW   NXLD      SET FIRST LOAD
31          LOX   ICNT
32          LDW   * 0
33          STW   COUNT     SET COUNT
34          CLR
```

35		STW	ERR	SET NO ERRORS
36	P01	LDX	COUNT	
37		DMS	1	THRU?
38		JMP	P02	NO
39		LDX	IERR	YES
40		LDW	ERR	
41		STW *	0	SET ERROR COUNT
42		SMB	R.RET	
43		JSX	R.RET	RETURN TO FORTRAN4
44		DATA	POOL	
45	P02	STX	COUNT	
46		LDX	NXLD	
47		LD8 *	0	
48		IXS	1	
49		NOP		JUST TO BE SAFE
50		STX	NXLD	
51		SMB	ESCDTCC1	
52		JSX	ESCDTCC1	
53		LDX	NXST	
54		STB *	0	
55		IXS	1	
56		NOP		TO BE SAFE
57		STX	NXST	
58		LDX	ERR	
59		IXS	1	
60	NXLD	DATA	0	
61		CLB	'*'	
62		SNE		ERROR?
63		STX	ERR	YES
64		JMP	P01	NO
65	*			
66	COUNT	DATA	0	
67	ERR	DATA	0	
68	NXST	DATA	0	
69	*			

70 POOL DATA 6.0.0
71 IBUF1 DATA 0
72 IBUF2 DATA 0
73 ICNT DATA 0
74 IERR DATA 0
75 END
BE

SUBROUTINE CONGRP

```
1 'CONVERT ML 13 CHAR. GROUPS TO BE USABLE
2 *
3 *   JANUARY 9, 1974
4 *
5 *   CALL FROM FORTRAN4 VIA:
6 *   CALL CONGRP(IBUF,IGRP,N,IFLAG)
7 *
8 *RESULT: 13 CHARACTER EBCDIC GROUPS STARTING WITH
9 *   THE NTH CHARACTER OF IBUF WILL BE
10 *   CONVERTED INTO 5 DISTINCT PARTS. IGRP
11 *   AND IFLAG WILL BE SET AS FOLLOWS:
12 *
13 *   IGRP(1) WILL BE SET TO A BINARY NUMBER
14 *   REPRESENTING THE CHARACTERS MAKING UP
15 *   THE MEASUREMENT SCANNER CHANNEL.
16 *   IGRP(2) WILL BE SET TO THE REMOTE
17 *   SCANNER CHANNEL REPRESENTING THE REMOTE
18 *   IF DATA WAS FROM A REMOTE; OTHER-
19 *   WISE, IGRP(2) WILL BE SET TO -1 (IF NOT
20 *   A REMOTE).
21 *   IGRP(3) WILL BE SET TO THE ASCII CHARACTER
22 *   REPRESENTING THE DVM FUNCTION. THIS WILL
23 *   NORMALLY BE AN ASCII M(X'0000'=-13056) OR
24 *   U(X'0600'=-19752) BUT CAN BE ANY OTHER
25 *   CHARACTER (ERROR=X'A400'=-22016)
26 *   IGRP(4) WILL BE SET TO A BINARY NUMBER REP-
27 *   RESENTING THE 6 CHARACTERS CONSISTING OF
28 *   POLARITY, OVER-RANGE FLAG, AND 4 VOLTAGE
29 *   DIGITS. IF THE OVER-RANGE FLAG IS NOT 0-2
30 *   IT WILL BE SET TO 2 AND BIT 10 (11TH BIT)
31 *   OF IFLAG WILL BE SET ON. IF POLARITY IS
32 *   OTHER THAN A + OR -, IT WILL BE ASSUMED
33 *   TO BE + AND BIT 9 (10TH BIT) OF IFLAG
34 *   WILL BE SET ON.
```

```

35 *      IGRP(5) WILL BE SET TO A BINARY NUMBER
36 *      REPRESENTING THE DUM SCALE CHARACTER.
37 *      IFLAG BITS 3-15 (4TH-16TH) WILL BE SET ON
38 *      IF THE NTH THRU N+12TH CHARACTERS ARE NOT
39 *      LEGAL OR APPROPRIATE. IF IFLAG=0, NO
40 *      ERRORS WERE DETECTED.
41 *

```

```

42 *  NOTE: IF AN ILLEGAL CHARACTER OTHER THAN MEN-
43 *        TIONED ABOVE, APPEARS IN ANY SUBGROUPING;
44 *        THE CORRESPONDING IFLAG BIT WILL BE SET
45 *        ON AND THE IGRP ELEMENT FOR WHICH IT
46 *        WOULD HAVE BEEN A PART WILL BE SET TO
47 *        X'8000' (-0).
48 *

```

```

49      LIBR  CONGRP
50      NTRY  CONGRP

```

```

51 *
52 *
53 *

```

```

54 GETT      PROC
55           LLB   P(1)
56           JSX   TRAN
57           BYTE  P(2),P(3)
58           ENDP

```

```

59 *
60 CALL      PROC
61           SMB   P(1)
62           JSX   P(1)
63           TRUE  P(0)>1
64           DATA P(2)
65           ENDC
66           ENDP

```

```

67 *
68 CONU      PROC
69      CALL  DECTOBIN,P(1)

```

70		JSX	CHEK	
71		STW	P(2)	
72		ENDP		
73	*			
74	CONGRP	DATA	POOL	
75		LDW	IBUF	GET BUF
76		SLL	1	MAKE BYTE
77		LDX	N	
78		ADD *	0	UP IT BY N
79		SUB	N1	TAKE ONE OFF
80		STW	BUFBLOC	
81		LDX	HOLDREF	
82		CLR		
83	P01	STB	INST0+1	
84		CLB	16	
85		SLE		THRU ZEROING?
86		JMP	P02	YES
87		CLR		NO
88	INST0	STW *	0	
89		LDB	INST0+1	
90		ADD	N1	
91		JMP	P01	
92	P02	GETT	0.0,'9'	
93		GETT	1.1,'9'	
94		GETT	2.2,'9'	
95		GETT	3.6,0	
96		GETT	4.7,0	
97		GETT	5.12,0	
98		CLB	'U'	
99		SEQ		
100		CLB	'M'	
101		LLB	1	
102		SEQ		
103		STB	HOLD5+6	
104		GETT	6.14,0	

105	CLB	'+'
106	SEQ	
107	CLB	'-'
108	SEQ	
109	LLB	'+'
110	STB	HOLD4
111	LLB	1
112	SEQ	
113	STB	HOLD5+7
114	GETT	7,15,'2'
115	CLR	
116	CMB	HOLD5+8
117	LLB	'2'
118	SEQ	
119	STB	HOLD4+1
120	GETT	8,16,'9'
121	GETT	9,17,'9'
122	GETT	10,18,'9'
123	GETT	11,19,'9'
124	GETT	12,20,'6'
125	CONU	HOLD1,HOLD1
126	HOLDREF EQU	\$-3
127	CONU	HOLD2,HOLD2+1
128	CALL	DECTOBIN,HOLD4
129	JSX	CHEK
130	LDX	IGRP
131	STW *	3 SET IGRP(4)
132	LDW	HOLD2
133	CMW	XABAO
134	LDW	HOLD2+1 GET REMOTE SUB-CHANNEL
135	SNE	
136	LDW	NM1 WAS NOT A REMOTE
137	STW *	1 SET IGRP(2)
138	LDW	HOLD1
139	STW *	0 SET IGRP(1)

140		LDW	HOLD3	
141		STW *	2	SET IGRP(3)
142		LDB	HOLD5	
143		AND	N15	
144		STW *	4	SET IGRP(5)
145		LDX	FLAGREF	
146		CLR		
147		STW	RET	
148	P03	STB	INSTF+1	
149		CLB	12	
150		SLE		THRU?
151		JMP	P04	YES
152		CLR		NO
153	INSTF	LDB *	0	
154		ORI	RET	
155		SLL	1	
156		STW	RET	
157		LDB	INSTF+1	
158		ADD	N1	
159		JMP	P03	
160	P04	LDW	RET	
161		SRL	1	
162		LDX	IFLAG	
163		STW *	0	SET IFLAG
164		CALL	R.RET,POOL	RETURN TO FORTRAN4
165	*			
166	TRAN	STX	RET	
167		STB	INST1+1	SET BUF LOAD
168		STB	INST3+1	SET FLAG LOC
169		LDW *	0	GET REFERENCES
170		STB	INST4+1	SET MAX NUMBER
171		SRL	0	
172		STB	INST2+1	SET SAVE LOC
173		LDX	BUFBLOC	
174	INST1	LDB *	0	GET DATA

175		CALL	EBCDTCC1	
176		LDX	HOLDBREF	
177	INST2	STB	* 0	SAVE ASCII
178		STB	SAV+1	
179	TP1	CLB	'*'	REMEMBER IF ERROR
180		LDX	FLAGREF	
181		LLB	1	
182		SNE		WAS IT ERROR?
183	INST3	STB	* 0	YES
184		CLR		NO
185		CMB	INST4+1	REMEMBER IF P(3)=0
186		LDW	SAV	
187		LDX	RET	
188		SNE		WAS P(3)=0?
189	TP2	JMP	* 1	YES
190	INST4	CLB	0	NO
191		SLE		IS DATA < OR = P(3)?
192		CLR		NO
193		CLB	'/'	YES
194		SLE		IS DATA A NUMBER?
195		JMP	TP2	YES
196		CLR		NO
197		STB	INST4+1	MAKE IT LOOK LIKE NO NUMBER
198		LLB	'*'	GET ERROR INDICATOR
199		JMP	TP1	
200	*			
201	CHEK	SEQ		
202		LDW	X8000	IF BAD
203		JMP	* 0	IF GOOD
204	*			
205	HOLD1	RES	3	
206	HOLD2	RES	3	
207	HOLD3	RES	1	
208	HOLD4	RES	3	
209	HOLD5	RES	7	

210	BUFBLOC	DATA	0
211	HOLDBREF	DATA	/HOLD1
212	FLAGREF	DATA	/HOLD5+1
213	SAU	DATA	0
214	RET	DATA	0
215	N1	DATA	1
216	N15	DATA	15
217	X8000	DATA	X'8000'
218	XA0A0	TEXT	' '
219	NM1	DATA	-1
220	*		
221	POOL	DATA	6,0,0
222	IBUF	DATA	0
223	IGRP	DATA	0
224	N	DATA	0
225	IFLAG	DATA	0
226		END	
BE			

SUBROUTINE CVPROC

```
S      LIBR CVPROC
      SUBROUTINE CVPROC(PAR,MPA,CLSTAB,MCL)
      LOGICAL FOUND
      INTEGER PAR(MPA),CLSTAB(MCL),DECRMT,PNTR,DEBUG,HASH
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      DATA NEXT/1/
      LEFT=MCL-NEXT+1
      NARG=PAR(1)-2
      IF (NARG.GT.LEFT) GO TO 600
C      IS THIS A DELETION, CREATION, OR AN EXPANSION.
      IF (PAR(3)) 300,100,200
C      CREATE A NEW CLASS VARIABLE.
100  CLSTAB(NEXT+1)=NARG
      DO 110 I=1,NARG
110  CLSTAB(NEXT+1+I)=PAR(3+I)
      CALL INHASH(PAR(2),-NEXT)
      CLSTAB(NEXT)=PAR(2)
      NEXT=NEXT+NARG+2
      CLSTAB(NEXT)=-1
      GO TO 900
C      EXPAND THE DEFINITION OF THE CLASS VARIABLE.
200  NDEX=-HASH(PAR(2),FOUND)
      IF (.NOT.FOUND) GO TO 610
      IF (NDEX+CLSTAB(NDEX+1)+2.NE.NEXT) GO TO 620
      DO 210 I=1,NARG
210  CLSTAB(NEXT-1+I)=PAR(3+I)
      NEXT=NEXT+NARG
      CLSTAB(NEXT)=-1
      CLSTAB(NDEX+1)=CLSTAB(NDEX+1)+NARG
      GO TO 900
C      DELETE A CLASS VARIABLE AND PACK THE CLASS TABLE.
300  NDEX=-HASH(PAR(2),FOUND)
      IF (.NOT.FOUND) GO TO 610
      CALL DEHASH(PAR(2))
      DECRMT=CLSTAB(NDEX+1)+2
      PNTR=NDEX
320  PNTR=PNTR+2+CLSTAB(PNTR+1)
      IF (PNTR.EQ.NEXT) GO TO 350
      CALL DEHASH(CLSTAB(PNTR))
      CALL INHASH(CLSTAB(PNTR),DECRMT-PNTR)
      GO TO 320
350  NEXT=NEXT-DECRMT
      DO 400 I=NDEX,NEXT
400  CLSTAB(I)=CLSTAB(I+DECRMT)
      GO TO 900
```

```

C   AN ERROR OCCURRED.  PRINT MESSAGE AND RETURN.
    600 CALL ERRPRT(20,17,NARG,LEFT,0,0,0)
        GO TO 900
    610 CALL ERRPRT(20,18,0,0,0,0,0)
        GO TO 900
    620 CALL ERRPRT(20,19,0,0,0,0,0)
C   DEBUGGING PRINTS FOLLOW.
    900 IF (DEBUG.NE.20) GO TO 950
        CALL OUTINT(GLSTAB,MCL,1,0)
        CALL OUTINT(NEXT,1,1,0)
        CALL OUTINT(NDEX,1,1,0)
    950 RETURN
        END
C;

```

```

S      LIBR DATTAP
      INTEGER FUNCTION DATTAP(PAR,MPA,OUTBUF,MOU,CLSTAB,MCL,WRTSWT,
*                                LREFC)
      LOGICAL WRTSWT,LREFC
      INTEGER PAR(MPA),OUTBUF(MOU),SUM,GSUM,FSTBLK(16),DEBUG,CLSTAB(MCL)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK4/IOCBD(33),IOCBN(33)
      DATA GSUM/0/
C      COMPUTE THE SUM OF THE STATUS FLAGS.
      SUM=0
      M6=MOU-6
      IERR=0
      INC=M6/2+6
      DO 50 I=1,NVAR
50     SUM=LOR(SUM,OUTBUF(INC+6+I))
C      IF SUM CHANGED SINCE LAST TIME, A STATUS FLAG CHANGED.  NOTE
C      THAT THE CONVERSE NOT NECESSARILY TRUE.  PRINT OUT THE STATUS
C      FLAGS IN THE CHANGED-SUM CASE.
      IF (SUM.EQ.GSUM) GO TO 70
      CALL OUTTXT('FLAG CHANGES IN:',8,3,0)
      CALL OUTINT(OUTBUF,5,4,1)
      CALL OUTTXT('PREV. SUM=',5,3,0)
      CALL OUTHEX(GSUM,1,2,0)
      CALL OUTTXT('CUR. SUM=',5,2,2)
      CALL OUTHEX(SUM,1,4,0)
      GSUM=SUM
C      SET UP THE FIRST RECORD OF THE DATA BLOCK TO BE PUT ON TAPE.
70     FSTBLK(1)=5
      FSTBLK(2)=IOCBD(27)
      FSTBLK(3)=NVAR
      FSTBLK(4)=M6
      FSTBLK(5)=OUTBUF(1)
      FSTBLK(6)=OUTBUF(2)
      FSTBLK(7)=OUTBUF(3)
      FSTBLK(8)=OUTBUF(4)
      FSTBLK(9)=OUTBUF(5)
      FSTBLK(10)=GSUM
      FSTBLK(11)=1
      IF (DEBUG.EQ.17) CALL OUTINT(FSTBLK,9,1,0)
C      PUT THE DATA BLOCK ON TAPE.
      CALL WRITMT(FSTBLK,16,-1,IERR,9)
      SUM=1
      IF (IERR.LT.0) GO TO 150
      CALL WRITMT(OUTBUF(7),M6,-1,IERR,9)
      IF (IERR.GE.0) GO TO 200
      SUM=2
150    CALL ERRPRT(17,16,SUM,NSCANS,OUTBUF(4),OUTBUF(5),0)
200    DATTAP=IERR
      RETURN
      END

```

C:

SUBROUTINE DEHASH

```
S      LIBR DEHASH
      SUBROUTINE DEHASH(KEY)
C  DELETES A KEY AND VALUE FROM THE HASH TABLE.
      LOGICAL FOUND
      INTEGER VALUES,KEY,KEYS,KEYSAV,KPLACE,DEBUG,HASH
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK2/KEYS(128),VALUES(128),KEYSAV,KPLACE
      K=HASH(KEY,FOUND)
      IF (.NOT.FOUND) GO TO 99
      IF (KEY.NE.KEYSAV) GO TO 99
      KEYS(KPLACE)=0
      VALUES(KPLACE)=0
      GO TO 900
      99 CALL ERRPRT(5,8,FOUND,0,0,0,0)
      900 RETURN
      END
```

C;

SUBROUTINE DRIVER

```
S      LIBR DRIVER
      SUBROUTINE DRIVER(FUNC,PAR,MPA,TABLE,MTA1,MTA2,OUTBUF,MOU,
*      CLSTAB,MCL)
      LOGICAL WRTSWT,LRECF
      INTEGER DAY,TIME,CLSVAR,DEBUG,FUNC,CLSTAB(MCL)
      INTEGER PAR,MPA,TABLE,MTA1,MTA2
      DIMENSION PAR(MPA),TABLE(MTA1,MTA2),NREC(2)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK3/DAY(2),TIME(2),CLSVAR
      COMMON /BLK4/IOCBD(33),IOCBN(33)

C      IOCBD(32)=0
      IOCBD(33)='LG'
      CALL OPENIT(IOCBD,L)
      CALL ADDRESS(OUTBUF,IOCBD(32))
      IOCBD(33)=MOU
      WRTSWT=.FALSE.
      LRECF=.FALSE.

C      SEARCH TIME TABLE FOR BOUNDS ON RECORD NUMBERS OF REQUESTED
C      RECORDS. THEN, CALCULATE EXACT RECORD NUMBERS FROM INFO THERE.
      J=1
      IF (DEBUG.EQ.15) CALL OUTINT(DAY,4,1,0)
      DO 120 I=1,MTA1
100    IF (TABLE(4,I).LT.DAY(J)) GO TO 120
      IF (TABLE(4,I).GT.DAY(J)) GO TO 110
      IF (TABLE(5,I).LT.TIME(J)) GO TO 120
110    NREC(J)=((DAY(J)-TABLE(1,I))*1440+TIME(J)-TABLE(2,I))/TABLE(6,I)
      NREC(J)=NREC(J)*TABLE(3,I)
      IF (DEBUG.EQ.15) CALL OUTINT(NREC,2,1,0)
      IF (J.EQ.2) GO TO 125
      J=2
      GO TO 100
120  CONTINUE

C      A NO MATCH OCCURRED. PRINT ERROR MESS AND RETURN.
      CALL ERRPRT(15,13,J,0,0,0,0)
      GO TO 200
125  L=0
      IOCBD(27)=NREC(1)
130  IF (L.GE.0) GO TO 140
      IF (L.EQ.-100) GO TO 130
      CALL ERRPRT(15,14,L,2,0,0,0)
      GO TO 200
140  CALL DSKRD(IOCBD,L)
150  IF (L.GE.0) GO TO 160
      IF (L.EQ.-100) GO TO 150
      CALL ERRPRT(15,14,L,1,0,0,0)
      GO TO 200
```

```

C   PASS CONTROL TO THE SUBROUTINE DESIRED.
    160 IERR=FUNC(PAR,MPA,OUTBUF,MOU,CLSTAB,MCL,WRTSNT,LRECF)
        IF (IERR.LT.0) GO TO 200
C   IF NECESSARY, WRITE OUT THE CURRENT RECORD BEFORE CONTINUING.
        IF (WRTSNT) CALL DSKUPD(IOCBD,L)
        IF (LRECF) GO TO 200
C   CHECK IF CURRENT RECORD IS LAST ONE DESIRED.
        IF (IOCBD(27).LT.NREC(2)) GO TO 130
        LRECF=.TRUE.
        GO TO 130
    200 CALL CLOSE(IOCBD,L)
        RETURN
        END
C:

```

SUBROUTINE DSKRTS

```
S      LIBR DSKRTS
      SUBROUTINE DSKRTS(OUTBUF,MOU,N,ICOM,IERR)
      INTEGER OUTBUF(MOU),DEBUG
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK4/IOCBD(33),IOCBN(33)
      IERR=0
      IOCBD(32)=0
      IOCBD(33)='LG'
      CALL OPENIT(IOCBD,L)
30    IF (L.GE.0) GO TO 40
      IF (L.EQ.-100) GO TO 30
      CALL ERRPRT(16,14,L,0,0,0,0)
      IERR=1
      GO TO 200
40    CALL ADRESS(OUTBUF,IOCBD(32))
      IOCBD(33)=MOU
      IF (DEBUG.EQ.16) CALL OUTHEX(IOCBD,33,1,0)
      GO TO (50,60,70),ICOM
50    IOCBD(27)=N-1
      CALL DSKRD(IOCBD,L)
      GO TO 80
60    CALL DSKWT(IOCBD,L)
      GO TO 80
70    CALL DSKUPD(IOCBD,L)
80    IF(L.GE.0) GO TO 200
      IF (L.EQ.-100) GO TO 80
      CALL ERRPRT(16,14,L,ICOM,0,0,0)
      IERR=1
200   IF (DEBUG.EQ.16) CALL OUTHEX(IOCBD,33,1,0)
      CALL CLOSE(IOCBD,L)
      RETURN
      END
```

C;

SUBROUTINE ERRPRT

```
S  LIBR ERRPRT
SUBROUTINE ERRPRT(ISUB,MESS,IARG1,IARG2,IARG3,IARG4,IARG5)
INTEGER DEBUG
COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
COMMON /BLK4/IOCBD(33),IOCBN(33)
DIMENSION NAMES(3,20)
DATA NAMES/'MA','IN',' ','GE','TC','OM','HA','SH',' ','
*IN','HA','SH','DE','HA','SH','IN','IT',' ','CL','AS','S ',
*CO','MI','NT','ER','RP','RT','RD','TA','PE','CO','NA',
*LL','PO','UT','BF','HE','XD','MP','UT','AB','LE','DR','IV',
*ER','DS','KR','TS','DA','TT','AP','EX',' ',' ','ST','DA','TA',
*CV','PR','OC'/

C  CALL OUTTXT('**ERROR-',4,3,0)
CALL OUTTXT(NAMES(1,ISUB),3,2,0)
CALL OUTTXT('TYPE',2,2,1)
CALL OUTINT(MESS,1,4,0)
CALL OUTINT(IARG1,1,3,0)
CALL OUTINT(IARG2,1,2,0)
CALL OUTINT(IARG3,1,2,0)
CALL OUTINT(IARG4,1,2,0)
CALL OUTHEX(IARG5,1,4,2)
IF (MESS.NE.14) GO TO 90
CALL OUTHEX(0,0,7,0)
DO 50 I=1,33
CALL OUTINT(I,1,2,0)
CALL OUTHEX(IOCBD(I),1,2,1)
IF (MOD(I,6).EQ.0) CALL OUTINT(0,0,5,0)
50 CONTINUE
CALL OUTINT(0,0,5,0)
90 RETURN
END
```

C;

SUBROUTINE EX

S LIBR EX
 SUBROUTINE EX
 INTEGER DEBUG
 COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
 COMMON /BLK4/IOCBD(33),IOCBN(33)
 CALL DELETF(IOCBD,L)
 CALL OUTTXT('EXIT',2,1,0)
 CALL EXIT
 RETURN
 END

C;

SUBROUTINE GETCOM

```

S      LIBR GETCOM
      SUBROUTINE GETCOM(COMMND,MCO,FCT,PAR,MPA,FLAG)
      INTEGER ARGCNT,STATE,CHCNT,FCT,CHCLS,VAL,PAR,TRANS,COMMND,CHAR
      INTEGER FLAG,CHARI,CODE,CLASS,HASH,DEBUG
      LOGICAL FOUND,SIGN
      DIMENSION TRANS(6,9),COMMND(MCO),PAR(MPA)
      COMMON /BLKI/NSCANS,NENT,DEBUG,NVAR
C      FINITE STATE TRANSITION TABLE.
      DATA TRANS/
1      2, 1, 1, 10, 1, 1,
2      2, 2, 3, 8, 10, 10,
3      4, 6, 7, 7, 9, 10,
4      4, 4, 5, 5, 10, 10,
5      4, 6, 7, 7, 9, 10,
6      10, 6, 7, 7, 10, 10,
7      4, 6, 7, 7, 9, 10,
8      10, 10, 10, 10, 10, 10,
9      10, 6, 10, 10, 9, 10/
C      A N , E S O
      CHARI=0
      ARGCNT=1
      STATE=1
      CHCNT=0
      FCT=0
      FLAG=0
      IF (DEBUG.EQ.2) CALL OUTTXT('CHAR CLASS STATE',9,1,2)
C      GET NEXT CHARACTER OF COMMAND INPUT.
50     CHARI=CHARI+1
      CHAR=COMMND(CHARI)
C      IGNORE BLANKS AND CHECK LEGALITY OF CHARACTER.
      IF (CHAR.EQ.37) GO TO 50
      IF (CHAR.GE.0.AND.CHAR.LE.41) GO TO 70
      CALL ERRPRT(2,2,CHARI,0,0,0,0)
      FLAG=1
      GO TO 999
C      GET CHARACTER CLASS AND STATE.
70     CHCLS=CLASS(CHAR)
      STATE=TRANS(CHCLS,STATE)
      IF (DEBUG.NE.2) GO TO 80
      CALL OUTINT(CHAR,1,3,0)
      CALL OUTINT(CHCLS,1,2,0)
      CALL OUTINT(STATE,1,4,1)
C      PASS OFF TO SECTION OF CODE DETERMINED BY CLASS AND STATE
C      IN THE TRANSITION TABLE.
80     GO TO (100,200,300,400,500,600,700,800,900,1000) STATE
C      CHARACTER RECEIVED WAS NOT A LETTER. IGNORE IT.
100    GO TO 50
C      PROCESS THE COMMAND CHARACTERS.
200    IF (CHCNT.GE.2) GO TO 50
      CHCNT=CHCNT+1
      FCT=FCT*36+CHAR
      GO TO 50

```

```

C   GEAR UP TO PROCESS THE ARGUMENTS OF THE COMMAND.
300 VAL=0
    CHCNT=0
    SIGN=.TRUE.
    ARGCNT=ARGCNT+1
    GO TO 50
C   PROCESS A VARIABLE ARGUMENT'S CHARACTERS.
400 IF (CHCNT.GE.2) GO TO 50
    CHCNT=CHCNT+1
    VAL=VAL*36+CHAR
    GO TO 50
C   IF FUNCTION NOT SV, DV, OR CV, RETRIEVE VARIABLE
C   ARGUMENT'S VALUE FROM THE HASH TABLE.
500 IF (FCT.EQ.1039.OR.FCT.EQ.499.OR.FCT.EQ.463) GO TO 700
    VAL=HASH(VAL,FOUND)
    IF (FOUND) GO TO 700
    ARGCNT=ARGCNT-1
    CALL ERRPRT(2,3,ARGCNT,0,0,0,0)
    FLAG=1
    GO TO 999
C   PROCESS A NUMERIC ARGUMENT'S CHARACTERS.
600 VAL=VAL*10+CHAR
    GO TO 50
C   END OF AN ARGUMENT BUILD-UP.  STORE IT.
700 IF (.NOT.SIGN) VAL=-VAL
    PAR(ARGCNT)=VAL
    IF (CHCLS.NE.4) GO TO 300
C   STORE NUMBER OF ARGUMENTS AND RETURN.
800 PAR(1)=ARGCNT-1
    GO TO 999
C   PROCESS A SIGN.
900 IF (CHAR.EQ.39) SIGN=.FALSE.
    GO TO 50
C   ERROR RETURN.
1000 CALL ERRPRT(2,4,0,0,0,0,0)
    FLAG=1
999 RETURN
    END
C;

```

SUBROUTINE HASH

```
S      LIBR HASH
      INTEGER FUNCTION HASH(KEY,FOUND)
C      HASH TABLE STORAGE ROUTINE. REF CACM,VOL.11,#1,P.43.
      LOGICAL FIRST,FOUND
      INTEGER WDSIZE,VALUES,KEYS,KEYSAV,KPLACE,DEBUG
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK2/KEYS(128),VALUES(128),KEYSAV,KPLACE
      DATA FIRST/.TRUE./
      DATA WDSIZE/16/-
      DATA N/7/

C      IF (DEBUG.NE.3) GO TO 20
      CALL OUTTXT('HASH KEY',4,3,0)
      CALL OUTINT(KEY,1,4,2)
20     IF (FIRST) GO TO 91
      1 IF (KEY.EQ.0) GO TO 98
      KEYSAV=KEY
C      USE PROD. OF KEY WITH APPROPRIATE MULTIPLIER AS HASH ADDRESS.
      KRAND=1
      IHASH=0
      KEYA=IABS(KEY)
      DO 11 I=1,WDSIZE,N
11     IHASH=IHASH+KEYA/(2**(I-1))
C      CHECK INDICATED PLACE IN TABLE TO SEE IF IT IS EMPTY,OCCUPIED,
C      BY THIS KEY,OR OCCUPIED BY ANOTHER KEY, WHICH WOULD REQUIRE
C      LOOKING FURTHER.
21     KPLACE=MOD(IHASH+KRAND/4,2**N)+1
      IF (KEYS(KPLACE).EQ.KEY) GO TO 31
      IF (KEYS(KPLACE).EQ.0) GO TO 41
      KRAND=MOD(5*KRAND,2**(N+2))
      IF (KRAND.EQ.1) GO TO 99
      GO TO 21
31     FOUND=.TRUE.
      HASH=VALUES(KPLACE)
      GO TO 999
41     FOUND=.FALSE.
      GO TO 999
91     K=2**N
      DO 92 I=1,K
92     KEYS(I)=0
      FIRST=.FALSE.
      GO TO 1
98     CALL ERRPRT(3,5,0,0,0,0,0)
      GO TO 41
99     CALL ERRPRT(3,6,0,0,0,0,0)
      GO TO 41
999    RETURN
      END
```

C;

SUBROUTINE HEXDMP

```
S      LIBR  HEXDMP
      SUBROUTINE HEXDMP(INBUF,MIN,IBEG,IEND)
      INTEGER DEBUG
      DIMENSION INBUF(MIN)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      M1=(IBEG+1)/2
      M2=(IEND+1)/2
      M=2*M1-1
10     M3=M1+12
      IF (M3.GT.M2) M3=M2
      CALL OUTINT(M,1,3,0)
      DO 20 I=M1,M3
20     CALL OUTHEX(INBUF(I),1,2,1)
      CALL OUTHEX(0,0,6,0)
      M=M+26
      M1=M1+13
      IF (M3.LT.M2) GO TO 10
      RETURN
      END
```

C;

SUBROUTINE INHASH

```
S      LIBR  INHASH
      SUBROUTINE INHASH(KEY,VALUE)
C  INSTALLS A NEW KEY AND VALUE IN HASH TABLE.
      LOGICAL FOUND
      INTEGER VALUE,VALUES,KEYS,KEYSAV,KPLACE,DEBUG,HASH
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK2/KEYS(128),VALUES(128),KEYSAV,KPLACE
      K=HASH(KEY,FOUND)
      IF (FOUND) GO TO 99
      IF (KEY.NE.KEYSAV) GO TO 99
      KEYS(KPLACE)=KEYSAV
      VALUES(KPLACE)=VALUE
      GO TO 900
99  CALL ERRPRI(4,7,FOUND,0,0,0,0)
900 RETURN
      END
```

C;

SUBROUTINE INIT

S

LIBR INIT

SUBROUTINE INIT(IFILE,MIF1,MIF2)

INTEGER FCN,HASH,IFILE

DIMENSION FCN(2,24),IFILE(MIF1,MIF2)

COMMON /BLK4/IOCBD(33),IOCBN(33)

DATA IOCBD/0,0,1,4*0,120,640,0,0,'NI','FX','UP','UF',18*0/

DATA IOCBN/0,0,1,4*0,120,640,0,0,'NI','FX','UP','UF',18*0/

DATA FCN/

1 1001,0100,

2 455,0200,

3 416,0300,

4 924,0400,

5 625,0500,

6 479,0600,

7 537,0700,

8 1028,0800,

9 1004,0900,

01 1039,1000,

11 499,1100,

12 859,1200,

13 1033,1300,

14 604,1400,

15 1180,1500,

16 1108,1600,

17 440,1701,

18 1071,1800,

19 1021,1903,

02 463,2000,

21 1022,2100,

22 1166,2200,

23 784,2300,

24 1000,2400/

C ALLOWABLE COMMANDS, LISTED IN ORDER OF APPEARANCE ABOVE, ARE:

C RT,CN,BK,PO,HD,DB,EX,SK,RW,SV,DV,NV,SP,GS,WS,US,C8,TR,SD,CV,SE,WE,

C LS,RS.

DO 20 J=1,24

CALL INHASH(FCN(1,J),FCN(2,J))

20 CONTINUE

IFILE(1,1)='TE'

IFILE(2,1)='FI'

IFILE(1,2)='CR'

IFILE(2,2)='AP'

CALL DSKINT(IOCBD,L)

CALL ADDRESS(IFILE(1,2),IOCBN(4))

CALL CREATE(IOCBN,L)

CALL DELETf(IOCBN,L)

CALL ADDRESS(IFILE(1,1),IOCBD(4))

40 IF (L.GE.0) GO TO 50

IF (L.EQ.-100) GO TO 40

CALL ERRPRT(6,14,L,0,0,0,0)

50 CALL CREATE(IOCBD,L)

60 IF (L.GE.0) GO TO 90

IF (L.EQ.-100) GO TO 60

CALL ERRPRT(6,14,L,0,0,0,0)

90 RETURN

END

C:

SUBROUTINES LSHIFT AND RSHIFT

```
1 *      SHIFT BUFFER LEFT AND RIGHT
2 *
3 *      NOVEMBER 10, 1974
4 *
5 *      CALL FROM FORTRAN4 VIA:
6 *      CALL LSHIFT(IBUF,ICNT,M,N)
7 *      OR
8 *      CALL RSHIFT(IBUF,ICNT,M,N)
9 *
10 *     WILL SHIFT CHARACTERS M THRU N OF IBUF
11 *     LEFT OR RIGHT ICNT POSITIONS. WILL
12 *     PLACE ZEROS IN POSITIONS SHIFTED OUT
13 *     OF AND THROUGH.
14 *
15 *****
16 **CAUTION**CAUTION**CAUTION**CAUTION**CAUTION**
17 *****
18 *     MEMORY BELOW IBUF WILL BE PROTECTED BY *
19 *     THIS PROGRAM; HOWEVER, IT IS LEFT TO THE *
20 *     USER TO INSURE THAT SHIFTS ARE NOT MADE *
21 *     IN OR OUT OF MEMORY LOCATED ABOVE IBUF. *
22 *****
23 *     NOTE: THIS IS THE EQUIVALENT OF A LOGICAL
24 *     BUFFER SHIFT BY BYTES LEFT OR RIGHT.
25 *
```

	26		LIBR	LSHIFT,RSHIFT	
	27		NTRY	LSHIFT,RSHIFT	
	28	*			
0000 0053	29	LSHIFT	DATA	POOL	
0001 804C	30		LDW	N1	
0002 2030	31		JSX	SETUP,N,M	
0003 0059					
0004 8058					
0005 9057	32		LDX	ICNT	
0006 B800	33		SUB *	0	
0007 100F	34		JMP	P01	
0008 0053	35	RSHIFT	DATA	POOL	
0009 8040	36		LDW	NM1	
000A 2030	37		JSX	SETUP,M,N	
000B 0058					
000C 8059					
000D 9057	38		LDX	ICNT	
000E A800	39		ADD *	0	
000F 7050	40	P01	STW	NXST	
0010 804F	41		LDW	NXLD	
0011 F051	42		CMW	LSLD	
0012 0800	43	SKIP	DATA	X'0800' THRU? (SLS OR SGR)	
0013 1021	44		JMP	P02	YES
0014 0130	45		CAX		NO
0015 8050	46		LDW	NXST	
0016 F052	47		CMW	MINLOC	
0017 5800	48		LDB *	0	
0018 9050	49		LDX	NXST	
0019 0890	50		SLE		OK TO STORE IT?
001A 3800	51		STB *	0	YES
001B 804F	52		LDW	NXLD	NO
001C A04E	53		ADD	TMP	

001D	704F	54	STW	NXLD	
001E	0140	55	CXA		
001F	A04E	56	ADD	TMP	
0020	100F	57	JMP	P01	
0021	8050	58	LDW	NXST	
0022	F051	59	CMW	LSLD	
0023	0000	60	SKIP2	DATA	X'0000' THRU ZEROING? (SLS OR SGR)
0024	1020	61	JMP	P04	YES
0025	F052	62	CMW	MINLOC	NO
0026	0130	63	CAX		
0027	0100	64	CLR		
0028	0090	65	SLE		OK TO SET A ZERO?
0029	3000	66	STB	* 0	YES
002A	0140	67	CXA		NO
002B	A04E	68	ADD	TMP	
002C	1022	69	JMP	P03	
* 002D	07FF	70	P04	SMB	R.RET
* 002E	2020	71	JSX	R.RET	
002F	0053	72	DATA	POOL	
0030	6050	73	SETUP	STX	NXST
0031	704E	74	STW	TMP	
0032	0640	75	LLB	X'40'	SLS FOR LSHIFT
0033	0010	76	SAP		
0034	0680	77	LLB	X'80'	SGR FOR RSHIFT
0035	3025	78	STB	SKIP+1	SET SKIP MODE
0036	3047	79	STB	SKIP2+1	
0037	9056	80	LDX	IBUF	
0038	0A31	81	SLL	D 1	
0039	0501	82	DXS	1	

003A	0A10	83	NOP	JUST TO BE SAFE
003B	0140	84	CXA	
003C	7052	85	STW	MINLOC
003D	9050	86	LDX	NXST
003E	9800	87	LDX *	0
003F	9800	88	LDX *	0
0040	A800	89	ADD *	0
0041	A04E	90	ADD	TMP
0042	7051	91	STW	LSLD
0043	B04E	92	SUB	TMP
0044	B800	93	SUB *	0
0045	9050	94	LDX	NXST
0046	9801	95	LDX *	1
0047	9800	96	LDX *	0
0048	A800	97	ADD *	0
0049	704F	98	STW	NXLD
004A	9050	99	LDX	NXST
004B	1802	100	JMP *	2
		101	*	
004C	0001	102	N1	DATA 1
004D	FFFF	103	NM1	DATA -1
004E	0000	104	TMP	DATA 0
004F	0000	105	NXLD	DATA 0
0050	0000	106	NXST	DATA 0
0051	0000	107	LSLD	DATA 0
0052	0000	108	MINLOC	DATA 0
		109	*	
0053	0006	110	POOL	DATA 6.0.0

0054	0000			
0055	0000			
0056	0000	111	IBUF	DATA 0
0057	0000	112	ICNT	DATA 0
0058	0000	113	M	DATA 0
0059	0000	114	N	DATA 0
		115		END

002E R.RET

NO ERRORS

ROUTINE MAIN

```
S      LIBR  MAIN
C      MAIN TEXT EDITOR PROGRAM.
      EXTERNAL DATTAP,STDATA
      INTEGER PAR,COMMND,FLAG,HASH,FCT,INBUF,OUTBUF
      INTEGER KEYS,VALUES,KEYSAV,KPLACE,DEBUG,SYMVAL,TABLE,SUBPAR
      INTEGER DAY,TIME,CLSVAR,FCSRES,CLSTAB
      LOGICAL FOUND
      COMMON FCSRES(100),OUTBUF(640),INBUF(2100),PAR(25),COMMND(74)
      COMMON IFILE(10,5),TABLE(6,25),CLSTAB(150)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK2/KEYS(128),VALUES(128),KEYSAV,KPLACE
      COMMON /BLK3/DAY(2),TIME(2),CLSVAR
      COMMON /BLK4/IOCBD(33),IOCBN(33)
      DATA DEBUG,NVAR/0,310/
      DATA MIN,MOU,MCO,MPA,MTA1,MTA2,MCL/2100,640,74,25,6,25,150/
      DATA MIF1,MIF2/10,5/

C      CALL OUTTXT('TOTAL ENERGY SYSTEM DATA EDITOR ',16,1,0)
      CALL INIT(IFILE,MIF1,MIF2)
50  CALL OUTTXT('?',1,1,0)
C  GO GET COMMAND INPUT AND TRANSLATE.
      CALL READIN(COMMND,N)
      CALL CODIT(COMMND,COMMND,N)
      COMMND(N+1)=41
      COMMND(N+2)=41
      DO 55 I=1,MPA
55  PAR(I)=0
C  GO DECODE THE COMMAND.
      CALL GETCOM(COMMND,MCO,FCT,PAR,MPA,FLAG)
      IF (FLAG.EQ.1) GO TO 50
      SYMVAL=HASH(FCT,FOUND)
      IF (FOUND) GO TO 60
      CALL ERRPRT(1,1,0,0,0,0,0)
      GO TO 50
C  BREAK OUT SUBROUTINE NUMBER AND PARAMETER NUMBER = POSITION
C  WHERE DAY AND TIME BEGIN. (IF = ZERO, NOT APPLICABLE.)
60  SUBPAR=MOD(SYMVAL,100)
      ISUB=SYMVAL/100
      IF (SUBPAR.EQ.0) GO TO 70
C  STORE DAY AND CONVERTED TIME. (HHMM GOES TO MMMM).
      DAY(1)=PAR(SUBPAR+1)
      DAY(2)=PAR(SUBPAR+3)
      TIME(1)=PAR(SUBPAR+2)-(PAR(SUBPAR+2)/100)*40
      TIME(2)=PAR(SUBPAR+4)-(PAR(SUBPAR+4)/100)*40
70  GO TO (100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,
      *1400,1500,1600,1700,1800,1900,2000,2100,2200,2300,2400),ISUB
```

```

C  PASS OFF TO THE REQUESTED ROUTINES.
100 CALL RDTAPE(INBUF,MIN,IERR)
   GO TO 50
200 CALL CONALL(INBUF,MIN,OUTBUF,MOU,IERR)
   GO TO 50
300 CALL BACKSP(PAR(2),PAR(3))
   GO TO 50
400 CALL POUTBF(OUTBUF,MOU,PAR(2),PAR(3))
   GO TO 50
500 CALL HEXDMP(INBUF,MIN,PAR(2),PAR(3))
   GO TO 50
600 DEBUG=PAR(2)
   GO TO 50
700 CALL EX
800 CALL SKIP(PAR(2),PAR(3))
   GO TO 50
900 CALL RNDIT(PAR(2))
   GO TO 50
1000 CALL INHASH(PAR(2),PAR(3))
   GO TO 50
1100 CALL DEHASH(PAR(2))
   GO TO 50
1200 NVAR=PAR(2)
   GO TO 50
1300 IOCBD(31)=PAR(2)
   GO TO 50
1400 CALL DSKRTS(OUTBUF,MOU,PAR(2),1,IERR)
   GO TO 50
1500 CALL UTABLE(OUTBUF,MOU,TABLE,MTA1,MTA2,IERR)
   IF (IERR.GT.0) GO TO 50
   CALL DSKRTS(OUTBUF,MOU,0,2,IERR)
   GO TO 50
1600 CALL DSKRTS(OUTBUF,MOU,0,3,IERR)
   GO TO 50
1700 CALL DRIVER(DATTAP,PAR,MPA,TABLE,MTA1,MTA2,OUTBUF,MOU,CLSTAB,MCL)
   GO TO 50
1800 CALL TRNRAW(INBUF,MIN,OUTBUF,MOU,TABLE,MTA1,MTA2,PAR(2),PAR(3))
   GO TO 50
1900 CALL DRIVER(STDATA,PAR,MPA,TABLE,MTA1,MTA2,OUTBUF,MOU,CLSTAB,MCL)
   GO TO 50
2000 CALL CVPROC(PAR,MPA,CLSTAB,MCL)
   GO TO 50
2100 CALL SEEKEF(PAR(2),IERR,PAR(3))
   CALL OUTINT(IERR,1,3,0)
   CALL OUTTXT('RECS SKIPPED',6,4,1)
   GO TO 50
2200 CALL WRITEF(3,PAR(2))
   GO TO 50
2300 CALL LSHIFT(INBUF,PAR(2),PAR(3),PAR(4))
   GO TO 50
2400 CALL RSHIFT(INBUF,PAR(2),PAR(3),PAR(4))
   GO TO 50
   END

```

C:

SUBROUTINES OUTHEX, OUTINT, AND OUTTXT

```
1  '      FORTRAN4 TEXT OUTPUT ROUTINES
2  *
3  *      FEBRUARY 24, 1975
4  *
5  *      CALLING SEQUENCES:
6  *      CALL OUTHEX(HEX,N,CODE,SKIP)
7  *              OR
8  *      CALL OUTINT(NUM,N,CODE,SKIP)
9  *              OR
10 *      CALL OUTTXT(IBUF,N,CODE,SKIP)
11 *
12 *      NUM = SINGLE INTEGER NUMBER TO BE
13 *            CONVERTED TO ASCII AND OUTPUT
14 *            AS A DECIMAL NUMBER.
15 *      HEX = SINGLE INTEGER NUMBER TO BE
16 *            CONVERTED TO ASCII AND OUTPUT
17 *            AS 4 HEX CHARACTERS.
18 *      SKIP= NUMBER OF SPACES TO OUTPUT BEFORE
19 *            OUTPUT OF TEXT OR 1ST HEX OR NUM.
20 *      IBUF= LOCATION OF FIRST WORD OF TEXT TO
21 *            BE OUTPUT.
22 *      N    = NUMBER OF TWO CHARACTER WORDS OF
23 *            TEXT TO BE OUTPUT, OR THE NUMBER
24 *            OF HEX OR NUMS TO BE CONVERTED
25 *            AND OUTPUT.
26 *      CODE= CARRIAGE CONTROL AS FOLLOWS:
27 *            0-DO NOTHING & RETURN IMMEDIATELY
28 *            1-(L/F)(TEXT/NUM)(C/R)
29 *            2-      (TEXT/NUM)
30 *            3-(L/F)(TEXT/NUM)
31 *            4-      (TEXT/NUM)(C/R)
32 *            5-(L/F)      (C/R)
33 *            6-      (C/R)
34 *            7-(L/F)
```

```

35 *          OTHER-DO NOTHING & RETURN IMMEDIATELY
36 *
37 *  ADDITIONAL COMMENTS:
38 *      NUM WILL BE RIGHT JUSTIFIED WITH SPACES
39 *      REPLACING LEADING ZEROS.
40 *      FIELD WIDTH OF NUM (INCLUDING LEADING
41 *      SPACES AND SIGN) WILL ALWAYS BE 6.
42 *      + SIGN WILL BE OUTPUT AS A SPACE.
43 *      - SIGN WILL BE OUTPUT AS A - SIGN AND
44 *      WILL BE PLACED IN THE POSITION IMMED-
45 *      IATELY TO THE LEFT OF THE MOST SIGNIFI-
46 *      CANT DIGIT OF THE NUMBER. HEX WILL
47 *      HAVE A FIELD WIDTH OF 4.
48 *      SKIP SPACES WILL BE PLACED BEFORE THE
49 *      TEXT OR BEFORE THE FIRST NUM OR HEX.
50 *      THE ABSOLUTE VALUE OF SKIP AND N WILL BE
51 *      USED AS THEIR DEFINED VALUE.
52 *  NOTE:
53 *      THE TEXT CONTAINED IN IBUF AND ALL OUTPUT
54 *      FROM EACH CARRIAGE RETURN WILL BE
55 *      IN BLOCKS OF EITHER 71 OR 72 IF N>72.
56 *      THAT IS, A C/R & L/F (IN THAT ORDER)
57 *      WILL BE ADDED BEFORE EACH 72'ND OR
58 *      73'RD CHARACTER. THIS WILL BE UPDATED
59 *      ON EACH CHARACTER AND WILL ONLY BE
60 *      RESET WHEN A C/R IS GIVEN FOR ANY
61 *      REASON.
62 *
63 *      ALL OUTPUT WILL BE ON LUN11 WHICH SHOULD
64 *      NORMALLY BE ASSIGNED TO THE CRT OR TTY.
65 *
66 *      LIBR  OUTHEX,OUTINT,OUTTXT
67 *      NTRY  OUTHEX,OUTINT,OUTTXT
68 *
69 LUN      EQU    11

```

70	DDIO	EQU	68
71	STAT	EQU	70
72	*		
73	SFOT	PROC	
74		DATA	P(1),P(2)+X'8000'
75		DATA	[0:7,LUN:5,14:4]
76		RES	3
77		DATA	X'8000'
78		RES	1
79		ENDP	
80	*		
81	DDIT	PROC	
82		STW	P(1)
83		CLR	
84		STW	P(2)+5
85		DDIO	P(2)
86		STAT	P(2)
87		ENDP	
88	*		
89	TRAN	PROC	
90		LDW	P(1)
91		STW	P(2)
92		ENDP	
93	*		
94	ADD0	PROC	
95		LDW	COUNTER
96		ADD	P(1)
97		ADD	P(1)
98		ENDP	
99	*		
100	DECR	PROC	
101		LDW	P(1)
102		SUB	P(2)
103		STW	P(1)
104		ENDP	

105	*		
106	SETT	PROC	
107		DATA	POOL
108		JSX	SETUP
109		ENDP	
110	*		
111	OUTHEX	SETT	
112	OUTHEX1	LDW	REPEAT
113		SAZ	
114		SAP	
115		JMP	GETOUT
116		LDX	BUFSU
117		LDW	* 0
118		SMB	HEXTCC
119		JSX	HEXTCC
120	NUMBUFR	DATA	NUMBUF
121		TRAN	N2, WCTMP
122		JSX	DOOUT
123		JMP	OUTHEX1
124	*		
125	OUTINT	SETT	
126	OUTINT1	LDW	REPEAT
127		SAZ	
128		SAP	
129		JMP	GETOUT
130		LDX	BUFSU
131		LDW	* 0
132		SMB	HEXTDEC
133		JSX	HEXTDEC, NUMBUF
134		SAM	
135		JMP	OUTINT4
136		LDX	NUMBUFR
137	OUTINT2	LDW	* 1
138		CLS	
139		SEC	

140	JMP	OUTINT3
141	IXS	1
142	JMP	OUTINT2
143	JMP	OUTINT2
144	OUTINT3	LLB 1
145	STB *	0
146	OUTINT4	TRAN N3, WCTMP
147	JSX	DOOUT
148	JMP	OUTINT1
149	*	
150	OUTTXT	SETT
151	TRAN	XAOAO, REPEAT
152	JSX	DOOUT
153	GETOUT	LDW CODESU
154	CLB	2
155	SEQ	
156	CLB	1
157	SEQ	
158	JSX	CR
159	JMP	RETURN
160	*	
161	DOOUT	STX DOOUTRET
162	LDW	REPEAT
163	CMW	ZERO
164	SNE	
165	JMP *	0
166	SAM	
167	JMP	DOOUT2
168	DOOUT1	LDX DOOUTRET
169	LDW	WCTMP
170	CMW	ZERO
171	SGR	
172	JMP *	0
173	LDW	COUNTER
174	CMW	N71

175		SLS	
176		JSX	CRLF
177		LDW	COUNTER
178		ADD	N1
179		SRL	1
180		STW	COUNTWD
181		LLB	36
182		SUB	COUNTWD
183		CMW	WCTMP
184		SLE	
185		TRAN	WCTMP,WC
186		DECR	WCTMP,WC
187		JSX	OUTFIOT1
188		JMP	DOOUT1
189	DOOUT2	STX	OFOT1RET
190		LDW	BUFSV
191		ADD	N1
192		STW	BUFSV
193		TRAN	NUMBUER,FIOT1
194		ADDD	WCTMP
195		CMW	N72
196		SLE	
197		JSX	CRLF
198		DECR	REPEAT,N1
199		TRAN	WCTMP,WC
200		JMP	OFOT1P1
201	*		
202	OUTFIOT1	STX	OFOT1RET
203	OFOT1P1	ADDD	WC
204		DOIT	COUNTER,FIOT1
205		LDW	FIOT1
206		ADD	WC
207		STW	FIOT1
208		LDX	OFOT1RET
209		JMP	* 0

210	*		
211	CR	CLR	
212		STW	COUNTER
213		LDW	X8D00
214		JMP	OUTONE
215	CRLF	STX	CRLFRET
216		JSX	CR
217		LDX	CRLFRET
218	LF	LDW	X008A
219	OUTONE	STX	OFOT2RET
220		DOIT	ONEBUF,FIOT2
221		LDX	OFOT2RET
222		JMP	* 0
223	*		
224	SETUP	STX	SETUPRET
225		LDX	CODE
226		LDX	* 0
227		DXS	5
228		JMP	SETUP01
229		IXS	4
230		JMP	RETURN
231		STX	CODESU
232		TRAN	IBUF,BUFSU
233		STW	FIOT1
234		LDX	N
235		LDW	* 0
236		SAP	
237		CMP	
238		STW	WCTMP
239		STW	REPEAT
240		LDX	SKIP
241		LDW	* 0
242		SAP	
243		CMP	
244		STW	SKIPCNT

245		LDW	CODESU
246		SAD	
247		JSX	LF
248	OUTSKIP	LDX	SETUPRET
249	SKIP01	LDW	SKIPCNT
250		CMW	N1
251		LDW	COUNTER
252		SNE	
253		JMP	SKIP02
254		SGR	
255		JMP	* 0
256		CMW	N71
257		SNE	
258		JMP	SKIP03
259		SLS	
260		JSX	CRLF
261		LDW	XAD00
262		JSX	OUTONE
263		DECR	SKIPCNT,N2
264		LDW	COUNTER
265		ADD	N2
266		JMP	SKIP04
267	SKIP02	CMW	N71
268		SLE	
269		JSX	CRLF
270	SKIP03	LOB	XAD00
271		SLL	8
272		JSX	OUTONE
273		DECR	SKIPCNT,N1
274		LDW	COUNTER
275		ADD	N1
276	SKIP04	STW	COUNTER
277		JMP	OUTSKIP
278	*		
279	SETUP01	OXS	3

280		JMP	RETURN
281	SETUP02	IMS	2
282		JMP	SETUP04
283		SXE	
284		JMP	SETUP03
285		JSX	CR
286		JMP	RETURN
287	SETUP03	JSX	LF
288		JMP	RETURN
289	SETUP04	JSX	CRLF
290	RETURN	SMB	R.RET
291		JSX	R.RET
292		DATA	POOL
293	*		
294	CRLFRET	DATA	0
295	DOOUTRET	DATA	0
296	OFOT1RET	DATA	0
297	OFOT2RET	DATA	0
298	SETUPRET	DATA	0
299	COUNTWO	DATA	0
300	COUNTER	DATA	0
301	SKIPCHT	DATA	0
302	NOTMP	DATA	0
303	REPSHT	DATA	0
304	ONEBUF	DATA	0
305	CODESU	DATA	0
306	NUMBUFSR	DATA	/HUMBUF
307	BUFSU	DATA	0
308	NC	DATA	0
309	ZERO	DATA	0
310	N1	DATA	1
311	N2	DATA	2
312	N3	DATA	3
313	N71	DATA	71
314	N72	DATA	72

315	XA0A0	DATA	X'A0A0'
316	X8000	DATA	X'8000'
317	X008A	DATA	X'008A'
318	NUMBUF	PES	3
319	*		
320	FIOT1	SFOT	0,NC
321	FIOT2	SFOT	ONEBUF,N1
322	*		
323	POOL	DATA	6,0,0
324	IBUF	DATA	0
325	N	DATA	0
326	CODE	DATA	0
327	SKIP	DATA	0
328	*		
329		END	
BE			

SUBROUTINE POUTBF

```
S      LIBR POUTBF
      SUBROUTINE POUTBF(OUTBUF,MOU,IBEG,IEND)
      INTEGER OUTBUF,DEBUG,SKIP
      DIMENSION OUTBUF(MOU)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
C      OUTPUT THE HEADER CHARACTERS.
      CALL OUTINT(OUTBUF,5,1,0)
      IF (IBEG.EQ.0) GO TO 90
C      OUTPUT THE MEASUREMENT VALUES NUMBERED IBEG TO IEND.
      N=1
      IF (IBEG.LT.IEND) N=N+IEND-IBEG
      CALL OUTINT(OUTBUF(IBEG+6),N,1,0)
90 RETURN
      END
C;
```

SUBROUTINE RDTAPE

```
S      LIBR  RDTAPE
      SUBROUTINE RDTAPE(INBUF,MIN,IERR)
      INTEGER INBUF,DEBUG
      DIMENSION INBUF(MIN)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      NCHAR=2*MIN
      IERR=0
      ICNT=0
      CALL READMT(INBUF,ICNT,NCHAR,IERR)
      IF (ICNT.GT.0.AND.IERR.EQ.0) GO TO 90
      CALL ERRPRT(10,9,ICNT,IERR,0,0,0)
      IERR=1
90     RETURN
      END
```

C;

```

1 *      READIN, CODE, AND DECODE ROUTINES
2 *
3 *      FEBRUARY 24, 1975
4 *
5 *      *****
6 *      FIRST SUBROUTINE:
7 *      THIS IS A LIBRARY PROGRAM TO READ THE
8 *      KEYBOARD INPUT DEVICE (DEFINED BELOW AS
9 *      KEYBORDI) INTO A PREVIOUSLY ESTABLISHED
10 *      FORTRAN BUFFER.
11 *
12 *      ENTRY IS BY THE FOLLOWING:
13 *      CALL READIN (IARG,N)
14 *      WHERE IARG IS THE FIRST LOCATION OF A
15 *      SINGLE INTEGER BUFFER IN WHICH TO PUT
16 *      THE ASCII DATA AND N WILL BE SET TO
17 *      THE NUMBER OF ASCII CHARACTERS READ
18 *      INTO THE BUFFER. NOTE: 2 ASCII
19 *      CHARACTERS WILL BE PLACED IN EACH
20 *      WORD OF BUFFER AND THE UNUSED
21 *      PORTION OF THE BUFFER (IF ANY) WILL
22 *      BE SET TO SPACES (UP TO 72).
23 *
24 *      NOTE: IBUF (AS LOCATED BY IARG) MUST BE AT LEAST
25 *      72 CHARACTERS (36 WORDS) IN SIZE.
26 *
27 *      *****
28 *      SECOND SUBROUTINE:
29 *      THIS ROUTINE WILL CODE AN ENTIRE ASCII
30 *      BUFFER INTO A SECOND BUFFER OF TWICE THE
31 *      SIZE OF THE ASCII BUFFER, HOWEVER THE TWO
32 *      BUFFERS MAY BE THE SAME.
33 *
34 *      CALL FROM FORTRAN4 VIA:

```

```

35 *      CALL CODIT (IBUF1,IBUF2,N)
36 *      WHERE IBUF1 IS THE LOCATION OF A BUFFER
37 *      CONTAINING ASCII DATA PACKED TWO TO A
38 *      WORD. IBUF2 IS THE LOCATION OF A BUFFER
39 *      IN WHICH TO PUT THE CODED DATA PACKED
40 *      ONE TO A WORD. AND N IS THE NUMBER OF
41 *      ASCII CHARACTERS TO BE CODED.
42 *
43 *  NOTE: IBUF2 MUST BE TWICE AS BIG AS IBUF1 OR IF
44 *  THEY ARE THE SAME, ONLY HALF OF THE BUFFER
45 *  CAN CONTAIN ASCII DATA.
46 *
47 *  CODES ARE:
48 *      0 THRU 9   ARE  0 THRU  9
49 *      A THRU Z   ARE 10 THRU 35
50 *      ,          IS  36
51 *      SPACE      IS  37
52 *      +          IS  38
53 *      -          IS  39
54 *      *          IS  40
55 *      $          IS  41
56 *      ALL OTHER ARE 42
57 *
58 * *****
59 *  THIRD ROUTINE:
60 *      THIS ROUTINE WILL CONVERT N CHARACTERS FROM
61 *      THE INTERNAL CODE ABOVE AND REPACK THEM AS
62 *      ASCII INTO ANOTHER (OR THE SAME) BUFFER.
63 *
64 *
65 *
66 *
67 *
68 *  CALL FROM FORTRAN4 VIA:
69 *      CALL DECDIT (IBUF1,IBUF2,N)

```

```

70 *      WHERE IBUF1 IS THE LOCATION OF A BUFFER
71 *      CONTAINING SINGLE WORD CODED CHARACT-
72 *      ERS, IBUF2 IS THE LOCATION OF A BUFFER
73 *      IN WHICH TO PLACE EQUIVALENT ASCII
74 *      CHARACTERS PACKED TWO TO A WORD, AND N
75 *      IS THE NUMBER OF CHARACTERS TO DECODE.
76 *
77 *      NOTE. IF N IS ODD, AN ADDITIONAL ASCII CHARACTER
78 *      WILL BE PLACED IN THE LAST HALF OF THE LAST
79 *      WORD. THIS CHARACTER WILL BE A SPACE.
80 *
81      LIBR  READIN,CODIT,DECDIT
82      NTRY  READIN,CODIT,DECDIT
83 *
84 DDIO     EQU    68
85 STAT     EQU    70
86 KEYBORDI EQU    10      LUN 10
87 *
88 READIN   DATA  POOL
89          LDW    IBUF1
90          STW    FIOT
91          SLL    1
92          ADD    N71
93          STW    LAST
94          DDIO   FIOT
95          STAT   FIOT
96          LDX    FIOT+4
97 P01      CXA
98          CMW    LAST
99          SLE
100         JMP    P02
101         LLB    ' '
102 X3800    STB * 0      SET SPACES
103         IXS    1
104         JMP    P01      IF IN UPPER MEMORY

```

105		JMP	P01	
106	P02	LDW	F10T+4	
107		SUB	IBUF1	
108		SUB	IBUF1	
109		LDX	IBUF2	
110	X7800	STW *	0	SET N (N IS ARG2 HERE)
111		JMP	RETURN	
112	*			
113	CODEIT	DATA	POOL	
114	CODE	EQU	CODEIT	
115		LDX	N	
116		LDW *	0	GET COUNT
117		ORI	X7800	
118		STW	STW1	SET STW * N (ORIG)
119	CODEIT	LDW	STW1	
120		AND	X07FF	
121		SUB	N1	
122		SAP		THRU?
123		JMP	RETURN	YES
124		ORI	X5800	NO
125		STW	LOB1	SET LOB * ? (WORK)
126		ORE	X2000	
127		STW	STW1	SET STW * ? (WORK)
128		LDX	IBUF1	
129		SLL	D 1	
130		CLR		
131	LOB1	LOB *	0	GET ASCII CHARACTER
132		CAX		
133		OXS	160	THIS PART OF CODEIT IS JUST
134		JMP	\$+2	TOO MUCH TO EXPLAIN WITH
135		JMP	GET42	COMMENTS
136		OXS	16	
137		JMP	P03	
138		IXS	16	
139	N6	DATA	6	

140		CXA		
141		STB	GETIT1+1	
142		CLR		
143		LDX	TAB1BR	
144	GETIT1	LDB	* 0	
145		JMP	P04	
146	P03	DXS	10	
147		JMP	#+2	
148		JMP	ADD10	
149		DXS	7	
150		JMP	#+2	
151		JMP	GET42	
152		DXS	26	
153	GET42	LDX	N6	
154		IXS	26	
155		NOP		MIGHT NOT SKIP
156	ADD10	IXS	10	
157	N1	DATA	1	WILL SKIP
158		CXA		
159	P04	LDX	IBUF2	
160	STWI	STW	* 0	SET CODED CHARACTER
161		JMP	CODEIT	CONTINUE ON
162	*			
163	DECDIT	DATA	POOL	
164	DECODE	EQU	DECDIT	
165		LDW	IBUF2	
166		SLL	1	
167		STW	IBUF2	MAKE IBUF2 BYTE
168		LDX	N	
169		LDW	* 0	GET COUNT
170		SAZ		
171		SAP		ANYTHING TO DO?
172		JMP	RETURN	NO
173		STW	LAST	YES-SET LAST REF
174		LDW	X3800	

175		STW	STBI	SET STB * 0
176	DECODEIT	LDW	STBI	
177		AND	X07FF	
178		SUB	LAST	
179		SAM		THRU?
180		JMP	P06	YES
181		ADD	LAST	NO
182		ADD	N1	
183		ORI	X3900	
184		STW	STBI	SET STB * ?
185		ORE	XA000	
186		STW	LDXI	SET LDX * ?
187		LDX	IBUF1	
188	LDXI	LDX	* 0	GET CODE
189		LLB	'?'	ASSUME BAD CODE
190		SXF		CODE?
191		JMP	P05	- & IS BAD
192		OXS	41	+ & MAY BE GOOD
193		IXS	41	
194		JMP	P05	BAD CODE
195		CXA		GOOD CODE
196		STB	GETIT2+1	
197		CLR		
198		LDX	TAB2BR	
199	GETITE	LOB	* 0	
200	P05	LDX	IBUF2	
201	STBI	STB	* 0	SET ASCII
202		JMP	DECODEIT	AND CONTINUE ON
203	P06	LDW	LAST	
204		SAC		
205		JMP	RETURN	
206		LDW	STBI	
207		ADD	N1	
208		STW	STBI2	
209		LLB	' '	

```

210 STB12      STB * 0          SET SPACE IF 000
211 RETURN     SMC R.RET
212           JSW R.RET
213           DATA POOL
214 *
215 N71         DATA 71
216 X07FF       DATA X'07FF'
217 X2000       DATA X'2000'
218 X5800       LDB * 0
219 XA000       DATA X'A000'
220 LAST        DATA 0
221 N36         DATA 36
222 TAB1BR      DATA /TAB1
223 TAB2BR      DATA /TAB2
224 FIOT        DATA 0,N36,C0:7,KEYBORDI:5,11:4]
225           RES 5
226 *
227 TAB1        BYTE 37,42,42,42,41,42,42,42
228           BYTE 42,42,40,38,36,39,42,42
229 *
230 TAB2        TEXT '0123456789ABCDEFGHIJKL'
231           TEXT 'MNOPQRSTUVWXYZ, +-*$'
232 *
233 POOL        DATA 5,0,0
234 IBUF1        DATA 0
235 IBUF2        DATA 0
236 N           DATA 0
237 *
238           END
BE

```

SUBROUTINE READMT

```

* READ ML TAPE INTO EBCD "SIZE" BUFFER
* (INTERRUPT VERSION)
*
* JULY 17, 1974
*
* ENTER FROM FORTRAN4 VIA:
* CALL READMT(IBUF,ICNT,NCHAR,IERR)
*
* WHERE: IBUF IS AN "NCHAR+NCHAR" WORD SIZE
* SINGLE INTEGER BUFFER.
* ICNT MUST CONTAIN AN EVEN NUMBER OF CHARAC-
* TERS TO SKIP BEFORE PUTTING THEM IN IBUF.
* ICNT WILL BE SET TO NEGATIVE COUNT IF THE
* RECORD STILL CONTAINS UNREAD DATA AND TO
* PLUS COUNT IF THE RECORD READ WAS
* COMPLETED. THE ABSOLUTE VALUE OF ICNT
* WILL EQUAL THE NUMBER OF CHARACTERS READ
* INTO IBUF. IF ICNT IS ZERO, AN END-OF-
* FILE WAS DETECTED.
* NCHAR MUST CONTAIN A POSITIVE EVEN INTEGER
* NUMBER EQUAL TO THE MAXIMUM EVEN NUMBER
* OF CHARACTERS TO BE READ INTO IBUF.
* IERR WILL BE SET TO THE NUMBER OF MAG-TAPE
* TYPE ERRORS (-1 IF A SINGLE ERROR APPLY-
* ING TO THE ENTIRE RECORD, -32767 IF THE
* RECORD WAS UNREADABLE, 0 IF NO ERRORS, OR
* TO THE NUMBER OF CHARACTERS IN POSSIBLE
* ERROR AS A RESULT OF MT-TYPE ERRORS).
*
* NOTE: IN NO CASE SHOULD NCHAR BE LARGER THAN
* TWICE THE NUMBER OF WORDS CONTAINED IN IBUF.
*
* LIBR READMT
* NTRY READMT
*
*
* D010 EQU 68
* STAT EQU 70
* MT02 EQU 2
* DR EQU 11
* EF EQU 10
* DRREF EQU DR+DR+DR+DR+1
* EFREF EQU EF+EF+EF+EF+1
* MTDEV EQU 0
* LISTLUN EQU 11
*
* GETV PROC
* TRUE MTDEV=0
* CLR
* ENDC
* FALSE MTDEV=0
* LDW DEV
* ENDC
* ENDP

```

```

*
READMT  DATA  POOL
        LDW    N4
        STW    ERTRY  SET ERROR TRY COUNTER
        LDW    NO
        STW    DRSAVE  SAVE D.R. LINK
        LDW * EFREF
        STW    EFSAVE  SAVE E.F. LINK
        JMP    P06
P01     LDW    JWAIT
        STW    WAIT
        LDW    ERCNT
        CMW    NO
        SNE
        JMP    P03      ERCNT=0
        SGR
        JMP    P02      ERCNT=-
        LDW    CHCNT    ERCNT=1 OR MORE
        SAP
        CMP
        CMW    ERCNT    FORCE CHCNT +
        SLE
        LDW    ERCNT    CHCNT>ERCNT (OK)
        SAZ
        SAP
        JMP    P03      - OK HERE (OR 0)
P02     LDW    ERTRY    STILL ERRORS
        DYS    1        TRIED 5 TIMES?
        JMP    P04      NO - TRY AGAIN
P03     LDW    IERR      YES-THEY MUST REALLY BE
        STW * 0          SET ERROR COUNT
        LDW    CHCNT
        LDW    ICNT
        STW * 0          SET CHARACTER COUNT
        LDW    NO
        LDW    DRSAVE
        STW * DRREF      REPLACE D.R. LINK
        LDW    EFSAVE
        STW * EFREF      REPLACE E.F. LINK
        SMB    R.RET
        JSX    R.RET      RETURN TO FORTRAN4
        DATA  POOL
P04     STW    ERTRY
P05     DIN    MTU9,0
        SAP
        JMP    P05      MT BUSY?
                        YES
        GETV
        DOT    MTU9,11    BACKSPACE
P06     DIN    MTU9,0
        SLL    1
        SAM
        JMP    P07      IS MT ON LINE?
                        YES
        DOIO    FIOT      NO
        STAT    FIOT

```

```

P07      GETV
        LDV      MTU9,0
        SAP      CONTROLLER READY?
        JMP      P07      NO
        SLL      1      YES
        SAP      DEVICE READY?
        JMP      P07      NO
        LDV      ICNT      YES
        LDW      * 0
        STW      SKIP      SET SKIP
        LDW      IBUF
        STW      DADR      SET BUFFER ADDRESS
        LDV      NO
        LDW      DR1REF
        STW      * DAREF      SET DATA LINK 1
        LDW      EF1REF
        STW      * EFREF      SET END-FN LINK 1
        CLR
        STW      ERFLG      SET NO LAST ERROR
        STW      ERCNT      SET NO ERRORS
        STW      CHCNT      SET NO DATA
        FALSE MIDEV=0
        LDW      DEV
        ENDC
        DOT      MTU9,2 READ 1 RECORD
        ENH      DR
        ENH      EF      GET SET UP TO WAIT
WAIT      JMP      WAIT      ON SOMETHING TO HAPPEN
*
DBL1      STW      ACR      SAVE ACR
        STV      IYR      SAVE IYR
        LDV      SKIP
        DKS      2      SAVE DATA?
        JMP      P08      NO
        LDV      NCHAR      YES
        LDW      * 0
        STW      SKIP      SET MAX BUF+2
        LDV      NO
        LDW      DR2REF
        STW      * DRREF      SET DATA LINK 2
        JMP      P09
P08      DIN      MTU9,15 GET DATA AND FORGET IT
        STV      SKIP      SET NEW SKIP
        JMP      P13
*
PRL2      STW      ACR      SAVE ACR
        STV      IYR      SAVE IYR
P09      DIN      MTU9,15 GET DATA
        LDV      SKIP
        DKS      2      IS BUFFER FULL?
        JMP      P10      NO
        LDV      CHCNT      YES
        SAM
        CMP
        STW      CHCNT      SET NOT THRU
        JMP      P11

```

P10	STX	SKIP	
	LDX	DADR	
	STW *	0	SAVE DATA
	IXS	1	
N0	DATA	0	
	STX	DADR	SET NEW BUFFER
	LDX	CHCNT	
	IXS	2	
N4	DATA	4	
	STX	CHCNT	INCREMENT COUNT
	DIV	MTU9,0	
	SAO		ERROR?
	JMP	P11	NO
	LDX	ERCNT	YES
	IXS	2	
ENTRY	DATA	0	
	STX	ERCNT	SET ERRORS UP BY ONE
	LDW	NM1	
	JMP	P12	
P11	CLR		
P12	STW	ERFLG	
P13	LDW	ACR	RECOVER ACR
	LDX	IXR	RECOVER IXR
	INR	DR	RETURN TO WHEREVER
*			
EFL1	DSB	DR	SHUT DOWN BOTH
	DSB	EF	INTERRUPTS
	STW	ACR	SAVE ACR
	STX	IXR	SAVE IXR
	LDW	CHCNT	
	SAZ		IS COUNT ZERO?
	JMP	P16	NO
	DIV	MTU9,0	YES
	SRL	4	
	SAO		END-OF-FILE?
	JMP	P14	NO
	LDW	SKIP	YES
	SAZ		
	SAP		
	JMP	P19	SKIP=0 OR LESS
	LDW	NM32767	SKIP=1 OR MORE
	JMP	P15	
P14	LDW	NM1	
P15	STW	ERCNT	
	JMP	P19	
P16	LDX	DADR	
	DXS	1	
	LDW *	0	
	CLB	0	REMEMBER LAST CHARACTER
	DIV	MTU9,0	
	SRL	1	
	SAO		RATE ERROR?
	JMP	P17	NO
	SEQ		YES-WAS LAST A ZERO?
	JMP	P14	NO
	JMP	P18	YES

P17	SEQ	WAS LAST A ZERO?
	JMP P17	NO
	LDW ERFLG	YES
	SAM	WAS LAST A ERROR?
	JMP P18	NO
	LDX ERCNT	YES
	SKM	
	DYS 2	
	STX ERCNT	SET ERROR COUNT
P18	LDX CHCNT	
	SXP	
	IXS 1	
	DYS 1	
	STX CHCNT	SET COUNT DOWN BY ONE
P19	LDW JP01	
	STW WAIT	SET EXIT LINK
	LDW ACR	RECOVER ACR
	LDX IXR	RECOVER IXR
	INR EF	RETURN TO WHENCE WE CAME

*

JP01	JMP	P01
JWAIT	JMP	WAIT
ERCNT	DATA	0
CHCNT	DATA	0
SKIP	DATA	0
DADR	DATA	0
DR1REF	DATA	DRL1
DR2REF	DATA	DRL2
EF1REF	DATA	EFL1
ERFLG	DATA	0
ACR	DATA	0
IXR	DATA	0
NM1	DATA	-1
NM32767	DATA	-32767
N2	DATA	2
DRSAVE	DATA	0
EFSAVE	DATA	0
	FALSE	MTDEV=0
DEV	DATA	MTDEV
		ENDC

*

FIOT	DATA	M4X8787,N2
	DATA	[0:7,LISILUN:5,14:4]
	RES	5

*

M4X8787	DATA	'M4',X'8787'
---------	------	--------------

*

P001	DATA	6,0,0
IBUF	DATA	0
ICNT	DATA	0
NCHAR	DATA	0
IERR	DATA	0
		EVD

SUBROUTINE SPLITA AND UNSPLT

```
1 *          SPLIT AND UNSPLIT INTEGERS
2 *
3 *          NOVEMBER 9, 1974
4 *
5 *          CALL FROM FORTRAN4:
6 *              CALL SPLITA(ARG1,ARG2,ARG3)
7 *              OR
8 *              CALL UNSPLT(ARG1,ARG2,ARG3)
9 *
10 *          ARG2=0,LEFT BYTE OF ARG1  (SPLITA)
11 *          ARG3=0,RIGHT BYTE OF ARG1  "
12 *              OR
13 *          ARG1=ARG2,ARG3              (UNSPLT)
14 *
15          LIBR  SPLITA,UNSPLT
16          NTRY  SPLITA,UNSPLT
17 *
0000 0017 18 UNSPLT  DATA  POOL
0001 901C 19          LDX   ARG3
0002 8800 20          LDW  * 0
0003 302D 21          STB   SAUE+1
0004 901B 22          LDX   ARG2
0005 8800 23          LDW  * 0
0006 0A18 24          SLL   8
0007 901A 25          LDX   ARG1
```

0008	1011	26		JMP	P1
		27	*		
0009	0017	28	SPLITA	DATA	POOL
000A	901A	29		LDX	ARG1
000B	8800	30		LDW *	0
000C	7016	31		STW	SAVE
000D	0A08	32		SRL	8
000E	901B	33		LDX	ARG2
000F	7800	34		STW *	0
0010	901C	35		LDX	ARG3
0011	502D	36	P1	LDB	SAVE+1
0012	7800	37		STW *	0
* 0013	07FF	38		SMB	R.RET
* 0014	2013	39		JSX	R.RET
0015	0017	40		DATA	POOL
		41	*		
0016	0000	42	SAVE	DATA	0
		43	*		
0017	0005	44	POOL	DATA	5.0.0
0018	0000				
0019	0000				
001A	0000	45	ARG1	DATA	0
001B	0000	46	ARG2	DATA	0
001C	0000	47	ARG3	DATA	0
		48	*		
		49		END	

0014 R.RET

NO ERRORS

SUBROUTINE STDATA

```
S      LIBR STDATA
      INTEGER FUNCTION STDATA(PAR,MPA,OUTBUF,MOU,CLSTAB,MCL,NRTSWT,
*      LREFC)
      LOGICAL NRTSWT,LREFC
      INTEGER PAR(MPA),OUTBUF(MOU),CLSTAB(MCL),VARNUM,PTR,BEG,END,DEBUG
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      STDATA=0
      NRTSWT=.TRUE.
C      CHECK IF VARIABLE TO BE CHANGED IS A SINGLE OR A CLASS VARIABLE.
      IF (PAR(3).LT.0) GO TO 100
      IF (PAR(3).EQ.0) GO TO 300
      IF (PAR(3).GT.NVAR) GO TO 300
C      PROCESS THE SINGLE VARIABLE.
      VARNUM=PAR(3)
      OUTBUF(VARNUM+6)=PAR(2)
      GO TO 900
C      PROCESS THE CLASS VARIABLE.
100  PTR=-PAR(3)
      BEG=PTR+2
      END=BEG+CLSTAB(PTR+1)-1
      DO 150 I=BEG,END
      VARNUM=CLSTAB(I)
      OUTBUF(VARNUM+6)=PAR(2)
150  CONTINUE
      GO TO 900
300  CALL ERHPRT(19,16,0,0,0,0,0)
900  RETURN
      END
```

C;

SUBROUTINE TRNRAW

```
S      LIBR TRNRAW
      SUBROUTINE TRNRAW(INBUF,MIN,OUTBUF,MOU,TABLE,MTA1,MTA2,NREC,KPRNT)
      INTEGER INBUF(MIN),OUTBUF(MOU),NREC,KPRNT
C      BEGIN LOOP TO READ, CONVERT, AND WRITE TO DISK.
      N=0
      DO 50 I=1,NREC
      CALL RDTAPE(INBUF,MIN,IERR)
      IF (IERR.GT.0) GO TO 90
      CALL CONALL(INBUF,MIN,OUTBUF,MOU,IERR)
      IF (IERR.GT.0) GO TO 90
      CALL UTABLE(OUTBUF,MOU,TABLE,MTA1,MTA2,IERR)
      IF (IERR.GT.0) GO TO 90
      CALL DSKRTS(OUTBUF,MOU,0,2,IERR)
      IF (IERR.GT.0) GO TO 90
      N=I
      IF (I.EQ.1) GO TO 20
      IF (I.EQ.NREC) GO TO 20
      IF (KPRNT.EQ.0) GO TO 50
      IF (MOD(I,KPRNT).NE.0) GO TO 50
C      OUTPUT MONITORING INFORMATION ON FIRST, LAST, AND
C      KPRNTTH RECORDS TRANSFERRED.
20    CALL OUTTXT('TRANSFER REC NO.',8,3,0)
      CALL OUTINT(N,1,4,0)
      CALL POUTBF(OUTBUF,MOU,0,0)
50    CONTINUE
90    CALL OUTINT(N,1,3,0)
      CALL OUTTXT(' RECS TRANSFERRED.',9,4,0)
      RETURN
      END
```

C;

SUBROUTINE UTABLE

```
S      LIBR UTABLE
      SUBROUTINE UTABLE(OUTBUF,MOU,TABLE,MTA1,MTA2,IERR)
      INTEGER DAY,TIME,CLSVAR,DEBUG
      INTEGER TABLE,OUTBUF,DELTAT
      DIMENSION OUTBUF(MOU),TABLE(MTA1,MTA2)
      COMMON /BLK1/NSCANS,NENT,DEBUG,NVAR
      COMMON /BLK3/DAY(2),TIME(2),CLSVAR
      DATA NSCANS,NENT/0,1/
      IERR=0
      NSCANS=NSCANS+1
C      CONVERT FROM HHMM TO MM.
      MINTIM=OUTBUF(5)-((OUTBUF(5)/100)*40
C      IF FIRST TIME THROUGH, GO INITIALIZE TABLE.
      IF (NSCANS.EQ.1) GO TO 900
      DELTAT=(OUTBUF(4)-TABLE(4,NENT))*1440+MINTIM-TABLE(5,NENT)
      IF (DELTAT.LT.0) GO TO 960
      IF (TABLE(6,NENT).EQ.0) GO TO 910
      IF (TABLE(6,NENT).EQ.DELTAT) GO TO 920
C      CHECK FOR TABLE FULL.
      IF (NENT.EQ.MTA1) GO TO 950
      NENT=NENT+1
C
900    DELTAT=0
      TABLE(1,NENT)=OUTBUF(4)
      TABLE(2,NENT)=MINTIM
      TABLE(3,NENT)=NSCANS-1
910    TABLE(6,NENT)=DELTAT
920    TABLE(4,NENT)=OUTBUF(4)
930    TABLE(5,NENT)=MINTIM
      IF (DEBUG.NE.14) GO TO 999
      CALL OUTINT(NENT,1,3,0)
      CALL OUTINT(TABLE(1,NENT),6,4,0)
      GO TO 999
950    CALL ERRPRT(14,12,MTA2,0,0,0,0)
      GO TO 998
960    CALL ERRPRT(14,20,MTA2,MTA2,MTA2,0,0,0)
998    IERR=1
999    RETURN
      END
```

C;

SUBROUTINE WRITMT

```
1 *      WRITE FROM CORE TO A DEVICE
2 *
3 *      JANUARY 17, 1975
4 *
5 *      FORTRAN4 CALLING SEQUENCE:
6 *      CALL WRITMT(IBUF,N,M,IERR,LUN)
7 *
8 *      IBUF IS THE BUFFER CONTAINING THE DATA TO
9 *      BE WRITTEN. IF M IS >0, THEN IBUF MUST
10 *      CONTAIN ASCII DATA; OTHERWISE, IBUF MAY
11 *      CONTAIN ANYTHING.
12 *      N IS THE NUMBER OF WORDS TO WRITE IF
13 *      M<0 OR M=0; OTHERWISE, N IS THE NUMBER
14 *      OF ASCII CHARACTERS TO WRITE (IF M>0).
15 *      IF M>0 AND N IS ODD, AN EXTRA SPACE WILL
16 *      BE WRITTEN ON THE MAG TAPE AS THE LAST
17 *      ASCII CHARACTER.
18 *      M IS A FORMAT CONTROL WORD AS FOLLOWS:
19 *      M=(-), BINARY FORMAT **NOTE:M ONLY HAS**
20 *      M=(0), SPECIAL FORMAT** REAL MEANING **
21 *      M=(+), ASCII FORMAT ** FOR 7-TRK MT **
22 *      IERR WILL BE SET (-) IF A NON-RECOVERABLE
23 *      MAG TAPE ERROR WAS ENCOUNTERED; IF THE
24 *      WRITE OPERATION WAS COMPLETED WITH NO
25 *      ERRORS, IERR WILL BE SET TO ZERO (0).
26 *      LUN IS THE LUN ON WHICH TO WRITE.
27 *      IF LUN IS MINUS, WORD 5 OF THE FIOT
28 *      WILL BE SET TO ZERO AND RETURN WILL
29 *      BE IMMEDIATE. THIS SIMULATES A REWIND
30 *      OF DISK FILE ZERO.
31 *
32 *      **MAG TAPE FORMAT**
33 *      7-TRACK:
34 *      M=0, ONLY THE FIRST 12 BITS OF EACH WORD IN
```

```

35 *      IBUF WILL BE WRITTEN ON THE TAPE. EACH
36 *      FRAME OF THE TAPE WILL CONTAIN 6 BITS OF
37 *      DATA BEGINNING WITH THE FIRST 6 BITS OF
38 *      IBUF(1) FOLLOWED BY THE 7TH-12TH BITS OF
39 *      IBUF(1) AND 1ST-6TH OF IBUF(2) AND SO ON
40 *      THRU THE 7TH-12TH BITS OF IBUF(N).
41 *      M=-, THE FIRST TWO FRAMES OF THE RECD WILL BE
42 *      ALL 1'S (12 BITS). EACH 4 FRAMES AFTER
43 *      THE FIRST 2 WILL CONTAIN A WORD OF 16 BIT
44 *      MEMORY AS FOLLOWS: (D=DATA BIT)
45 *      DDDDDD DDDDDD DDDDDD 000000
46 *      M=+, EACH MAG TAPE FRAME WILL CONTAIN ONE
47 *      ASCII CHARACTER WITH ITS FIRST TWO BITS
48 *      STRIPPED OFF.
49 *      IE: SPACE=X'A0'=10100000=100000
50 *      COMMA=X'AC'=10101100=101100
51 *      1=X'B1'=10110001=110001
52 *      N=X'CE'=11001110=001110
53 *      W=X'D7'=11010111=010111
54 *      ETCETERA
55 *      9-TRACK:
56 *      MAG TAPE FRAMES WHEN TAKEN TWO AT A TIME WILL
57 *      BE A MIRROR IMAGE OF A WORD OF MEMORY.
58 *
59 *      NOTE: NO PARALLEL PROCESSING MAY TAKE PLACE WHEN
60 *      THE PROCESSOR IS OPERATING WITHIN THIS SUB-
61 *      ROUTINE. THIS SUBROUTINE WILL TAKE CARE OF
62 *      NOT ALLOWING IT, THEREFORE THE USER OF THIS
63 *      PROGRAM NEED NOT WORRY ABOUT IT.
64 *
65 *      LIEB  WRITMT
66 *      NTRY  WRITMT
67 *
68 DOIO      EQU    68
69 STAT      EQU    70

```

```

70 *
71 WRITMT DATA POOL
72 LDX LUN
73 LDW * 0 GET LUN
74 SAM REWIND THE DISK?
75 JMP NODSKRWD NO
76 CLR YES
77 STW FIOT+5
78 JMP RETURN
79 NODSKRWD SLL 8
80 LLB X'E0'
81 SRL 4
82 STW FIOT+2
83 LDW IBUF
84 STW FIOT SET BUF LOC
85 LDX N
86 CLR
87 CMW * 0 REMEMBER MODE
88 STW SAVE ASSUME NOT ASCII
89 LDX N
90 LDW * 0 COUNT TO ACR
91 SLS MODE?
92 JMP P01 NOT ASCII
93 ADD FIOT ASCII
94 ADD FIOT CALC LAST BYTE LOC
95 CAX
96 STW LASTB SAVE LAST BYTE LOC
97 LDW SIGB GET TRIGGER BIT
98 LDB * 1 GET LAST BYTE+1
99 STW SAVE SAVE LAST BYTE+1
100 LLB ' '
101 STB * 1 SET LAST BYTE+1 TO SPACE
102 LDX N JUST IN CASE N IS ODD.
103 LDW * 0 GET ASCII COUNT
104 ADD N1 BUMP IT BY ONE AND

```

105		SRL	1	FORCE IT EVEN FOR WC
106	P01	STW	WDCNT	SET WORD COUNT
107		CLR		ASSUME NOT SPECIAL
108		SNE		MODE?
109		LDW	SIGB	SPECIAL
110		STW	FIOT+6	NOT SPECIAL-SET FORMAT
111		LDW	WCLOC	ASSUME ASCII
112		SLE		MODE?
113		ORI	SIGB	BINARY
114		STW	FIOT+1	ASCII OR SPECIAL
115		DOIO	FIOT	WRITE THE BUFFER ON MT
116		STAT	FIOT	WAIT ON IT TO FINISH
117		LDW	SAVE	
118		SAM		DID WE CHANGE BUFFER
119		JMP	P02	NO
120		LDX	LASTB	YES
121		STB	* 1	REPLACE LAST BYTE+1
122	P02	LDX	FIOT+3	MT STAT TO IXR
123		CLR		ASSUME NO ERROR
124		SXE		ERRORS?
125		LDW	SIGB	YES
126		LDX	IERR	NO
127		STW	* 0	SET ERROR CODE
128	RETURN	SMB	R.RET	
129		JSX	R.RET	RETURN TO FORTRAN4
130		DATA	POOL	
131	*			
132	SAVE	DATA	0	
133	LASTB	DATA	0	
134	SIGB	DATA	X'8000'	
135	WDCNT	DATA	0	
136	WCLOC	DATA	WDCNT	
137	H1	DATA	1	
138	*			
139	FIOT	RES	8	

```
140 *  
141 POOL      DATA  7.0,0  
142 IBUF      DATA  0  
143 N         DATA  0  
144 M         DATA  0  
145 IERR      DATA  0  
146 LUN       DATA  0  
147 *  
148          END  
BE
```


APPENDIX B

APPENDIX B: FORMAT OF RAW DATA INPUT TAPE

DENSITY: 800 f.p.i.

PARITY: Odd.

TRACKS: 9.

FORMAT: EBCDIC Characters.

LOGICAL RECORD LENGTH: $19 + 13 * NVAR$, where $NVAR$ = the number of voltage recording devices scanned by the DAS.

BLOCKING FACTOR: 1.

NUMBER OF LOGICAL RECORDS: Variable

RECORD INTERPRETATIONS:

POSITIONS

MEANING

1 - 12	twelve digits used as a header,
13 - 15	day of the year,
16 - 17	hour of the day,
18 - 19	minutes of the hour,
20 - 24	scanner channel number,
25	voltmeter function (M = millivolts, V = volts),
26	polarity (+ or -),
27	over-range flag (0,1, or 2),
28 - 31	voltage reading,
32	characteristic for voltage reading.

APPENDIX C

C-0

APPENDIX C: FORMAT OF THE 1108 COMPATIBLE OUTPUT TAPE

The following constitute tentative file formats for the RAYTHEON generated UNIVAC 1108 compatible magnetic tape file. They are tentative in that not all block types have been completely defined, and thus their representation has been left open.

The tape file created by the Data Editor will be used to convey edited data, comments inserted under control of the Editor, structured textual information, transducer parameter updates, data order index maps, and any other information required that will help to make the data tape self documenting.

Data scans will be in order by time with the other types of information inserted at that point in time with which they are concerned. Each data scan, or other block of information, will be written out as two physical records on the tape. The first record will have a fixed format and length (currently 16 words) for all block types. The second record will vary in length and format depending on the block type. Because the physical format of the type is dependent upon 7-track versus 9-track and other representational variations (see documentation for subroutine WRITMT), we give here only the logical format. Numeric values will normally be representable by 16 bits per character (dependent upon 7-track versus 9-track). All data values will be in binary.

The following defines the first record:

<u>RAYTHEON WORD</u>	<u>16 BIT CONTENTS</u>
1	Block Type
2	Sequence Number
3	Number of channels currently assumed
4	Length of second record of block
5-7	Twelve digit header stored in 3I4 format
8	Calendar day of year
9	Time to nearest minute in HHMM format
10	Condition code check sum
11	Binary/formated switch
12-16	Available for future use.

Codes for the Block type are as follows:

1. Unstructured text
2. Structured text
3. Index mapping function
4. Transducer parameter update
5. Data scan

For the second record, we give here only the format for the data scan block type. The second record of a data scan block will be 634 words long. The first 317 words contain the scan data, while the second 317 words contain the corresponding status codes. The sequence order for the scan data will probably be changed from time to time, but initially will be (for CEB Channels only) that in which Channel I is stored in the I^{TH} location, $10 \leq I \leq 180$. The remote channels may also be stored in a similar manner by breaking the DAS code into two indices.

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET		1. PUBLICATION OR REPORT NO. NBSIR 75-735	2. Gov't Accession No.	3. Recipient's Accession No.
4. TITLE AND SUBTITLE The Total Energy Data Editor			5. Publication Date June 1975	
			6. Performing Organization Code	
7. AUTHOR(S) Richard H. F. Jackson			8. Performing Organ. Report No.	
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234			10. Project/Task/Work Unit No.	
			11. Contract/Grant No.	
12. Sponsoring Organization Name and Complete Address (Street, City, State, ZIP) Department of Housing and Urban Development Office of Policy Development and Research Washington, D. C. 20410			13. Type of Report & Period Covered Final	
			14. Sponsoring Agency Code	
15. SUPPLEMENTARY NOTES				
16. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.) This report documents the Total Energy Data Editor, a computer program developed to process the data to be collected by the ongoing Total Energy Project at the National Bureau of Standards. Consisting of a mix of FORTRAN and RAYTHEON machine language subroutines, the Editor is a powerful, interactive program written to be run on a Raytheon 704 minicomputer with two tape drives and a disk pack. Since this document is also meant as a user's manual, it includes a dictionary of commands, complete discussions and listings of individual subroutines, as well as an explanation of the workings of the program.				
17. KEY WORDS (six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons) Data editing; computers; energy conservation				
18. AVAILABILITY ☒ Unlimited ☐ For Official Distribution. Do Not Release to NTIS ☐ Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. CI3 ☐ Order From National Technical Information Service (NTIS) Springfield, Virginia 22151		19. SECURITY CLASS (THIS REPORT) UNCLASSIFIED		21. NO. OF PAGES
		20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED		22. Price

